LEVEL II

ampus
omputing
etwork

AD A103394

DTIC
ELECTE
AUG 2 1981
S D
H

AN IP SERVER FOR NSW

niversity of
alifornia,
os
ngeles

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

81 8 26 084

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: (14) CCN/TR7 — 2. GOVT ACCESSION NO.: AD-A103	3. RECIPIENT'S CATALOG NUMBER: 394
4. TITLE (and Subtitle): AN IP SERVER FOR NSW.	5. TYPE OF REPORT & PERIOD COVERED: Semi-annual Technical Report, 6/1/75 ~~- 11/30/75~~
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): R. T./Braden, H. C./Ludlam	8. CONTRACT OR GRANT NUMBER(s): MDA 903-74-C-0083
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Campus Computing Network, UCLA C0012, Los Angeles, California 90024	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Program Code 4P10, ARPA Order -2543/3
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209	12. REPORT DATE: April 1976
	13. NUMBER OF PAGES: 90
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): NONE
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited

DTIC

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

ARPANET protocol, National Software Works, Batch Tool-Bearing Host, generalized Remote Job Entry protocol, file transfer, IBM 360 implementation, program logic.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

-SEE OVER-

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DD FORM 1 JAN 73 1473 A EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410283

This report describes an ARPANET remote job entry facility developed by UCLA-CCN as part of the National Software Works ("NSW") project, making the CCN 360/91 a "batch tool-bearing host" for NSW. The implementation of this facility is based on a server process for the interim NSW protocol IP, which was originally designed for Tenex-to-B4700 batch job control. The main component of the IP Server at CCN, the Message Processor or "MP", is largely written in PL/1 and runs as a "pseudo-user" under TSO.

University of California at Los Angeles
405 Hilgard Avenue,
Los Angeles, California 90024

CAMPUS COMPUTING NETWORK

Semiannual Technical Report

AN IP SERVER FOR NSW

R. T. Braden
H. C. Ludlam

CCN/TR7

April 1, 1976

This work was sponsored by the Advance Research Projects Agency of the Department of Defense, under Contract Number MDA903 74C 0083, Order 2543/3.

AN IP SERVER FOR NSW
April 1, 1976 -- CCN/TR7

REPORT SUMMARY

This report describes an ARPANET remote job entry facility developed by UCLA-CCN as part of the National Software Works ("NSW") project, making the CCN 360/91 a "batch tool-bearing host" for NSW. The implementation of this facility is based on a server process for the interim NSW protocol IP, which was originally designed for Tenex-to-B4700 batch job control. The main component of the IP Server at CCN, the Message Processor or "MP", is largely written in PL/1 and runs as a "pseudo- user" under TSO.

The evolution of the IP protocol is discussed in Section 1, and the protocol is fully defined in Section 2. Section 3 gives a general description of the CCN IP Server implementation, while the program logic of MP is described in detail in an appendix.

Implementation of this IP Server program has allowed the 360/91 to provide a useful computing capability for NSW users. It will provide experience with batch job control under NSW and an interim operating capability in advance of the specification and implementation of the full NSW mechanism -- MSG, File Package, and Foreman.

1. INTRODUCTION

1.1. UCLA/CCN AND NSW

The UCLA Campus Computing Network is the general-purpose central computing facility for UCLA, operating an IBM 360 Model 91 CPU with 4 million bytes of main memory. Software service systems on this machine include (Ref. 1): batch-processing under the IBM operating system OS/MVT, a fast-batch subsystem called QUICKRUN, the APL time-sharing system, IBM's general- purpose time-sharing system TSO, and a display-based conversational remote-job-entry system called URSA.

User access to CCN is via self-service unit-record equipment locally, medium-speed binary synchronous remote job entry ("rje") terminals, ASCII and IBM 2741 typewriter terminals for TSO and APL, IBM 3270 and CCI CC301 display terminals for URSA and TSO, and the ARPANET.

CCN has provided service to ARPANET users since 1971, using the following Network protocols:

* A "private" Network-rje protocol called NETRJS;

* Server-TELNET access to TSO;

* an FTP Server.

The National Software Works ("NSW") is attempting to create a centralized accounting, management, and file system as well as a uniform flavor of command language spanning a number of server host computers on the ARPANET. The NSW is intended to give government application programmers ready access to the rich variety of software tools which are available, or potentially available, on ARPANET server hosts. The NSW access control, file cataloging, and accounting functions are centralized in a program called the Works Manager, or "WM".

NSW server hosts are called Tool-Bearing-Hosts or "TBH"'s in NSW parlance. As part of the overall NSW effort, the UCLA Campus Computing Network is charged with making its IBM 360/91 a TBH. The particular strengths of the CCN machine are (1) its large batch-processing computational power, and (2) the large number of tools written for the IBM 360 systems. As a result, first priority for the NSW effort at CCN has been assigned to making the 360/91 a batch TBH; the implementation of interactive tools under TSO has been deferred until later.

During the period covered by this report, CCN has developed and demonstrated an interim batch TBH capability. This report describes this implementation, which is basically a server for the Network IP protocol. The history and development of the IP protocol are described in the remainder of Section 1. Section 2 contains a complete definition of IP as implemented between the prototype WM and the 360/91 at CCN. Section 3 describes the program organization of the CCN implementation.

1.2. BATCH SUBMISSION UNDER NSW

Under NSW, a "batch tool" is one which is executed in the background, leaving a foreground (interactive) user free for other work (see Ref. 2). Furthermore, for efficiency many batch hosts (including CCN) require that input files for batch tools be "pre-staged". That is, before a batch job can be "submitted", i.e. added to the batch processing queue of a batch host, all of its input files must be made resident in that host's local file space.

To cause a batch tool to be executed, an NSW user will interact with a program in the Works Manager called the Interactive Batch Specifier, or IBS. It is the function of IBS to query the user for the names of all input files, resource limits, etc., and to build tables describing the job. When the job description is complete the IBS passes the tables to the WM and exits, allowing the user to enter another interactive program while WM runs the job in the background. The user can query the WM for the status of his jobs, or the WM may send him a message when output from a job is available for viewing from his console.

Batch job execution under NSW is actually controlled by a WM process called the Works Manager Operator, or "WMO". When a new batch job is specified by a user, the WM passes the description of the job to the WMO, which maintains tables of all batch jobs currently in progress. The WMO invokes primitive operations at the batch TBH's to cause these jobs to be processed.

In order to fully function as a batch TBH, a host will be required to implement primitive operations which are conceptually equivalent to the following:

* File Transfer operations --

Send a file over the Network to the TBH:

```
SEND(<local file name>, <file> ) ;
```

GET a file from the TBH over the Network:

```
GET( <local file name>-> <file> ) ;
```

Create a new file on the TBH;

```
CREATE(<file attributes> -> <local file name> ) ;
```

Delete a file on the TBH:

```
DELETE( <local file name> );
```

Check existence and attributes of a file:

```
FIND( <local file name> -> <file attributes>) ;
```

Rename a file on the TBH:

```
RENAME( <old local name>, <new local name> ) ;
```

Local file copy:

```
LOCALCOPY( <local file name >,<local file name> ) ;
```

* Job Submission and Deletion --

Submit a Job, i.e. cause a specified local file to be interpreted as a job-control stream by the TBH operating system:

```
SUBMITJOB( <local file name> -> <job name> ) ;
```

Delete job from the tables and queues of the batch system.

```
DELETEJOB( <job name> ) ;
```

* Job Completion Signals and Status Query --

Spontaneously notify the WM when batch job output is ready for retrieval;

```
-> <status=done>
```

Answer queries on job status:

STATUS(<jobname> -> <status>) ;

We can use a traditional remote job entry scenario as an example to illustrate the use of these primitive operations by the WMO. Suppose a batch job is to be run with one input file containing both job-control commands and data, and one output file. The WMO could use the following sequence:

CREATE(<input file attributes> -> <input file name>) ;

CREATE(<output file attributes> -> <output file name>);

The WMO would use the <input file name> and <output file name> returned by the TBH to complete the JCL in <input file> in a TBH-dependent manner. Then it would continue:

SEND(<input file name>, <input file>) ;

SUBMITJOB(<input file name> -> <jobname>) ;

The WMO may periodically poll for completion by sending STATUS messages, or may await a spontaneous Job-Done message from the TBH. In either case, when the output is available it executes:

GET(<output file name> -> <output file>) ;

DELETE(<input file name>) ,

DELETE(<output file name>) ;

DELETEJOB(<job name>) ;

The NSW batch primitives listed earlier allow much more complex scenarios than this. For example, many batch tools will need multiple input and/or multiple output files. Often only one of the input files will vary from one run to the next, so Network usage and delays can be significantly decreased by keeping local copies of the invariant files at the batch host. Similarly, the user may not want to see all of the output files from a particular tool, and those he doesn't want need not be transmitted across the Network. In summary, the NSW primitives for batch TBH are more general than those of a traditional rje server, in that NSW logically separates the file-transfer functions from job submission/deletion functions.

1.3. THE EVOLUTION OF IP

Network IP was designed as a protocol for the WMO to use to request primitive operations of a batch TBH. IP allows WMO to invoke all of the primitives listed above except local copy. It was felt that local copy could be omitted from the interim protocol, as real batch tools seldom clobber their input files (and in fact, the local operating system may enforce read-only access under control of the JCL). The interim batch TBH service at CCN is based upon an implementation of a server for the IP protocol. Figures 1-3 illustrate the three stages in the evolution of IP for this purpose.

IP was originally designed by Tom Hamrick of Science Applications, Inc., and by Charles Muntz of Massachusetts Computer Associates, in order to make the Burroughs B4700 at Gunter AFS a batch TBH. As shown in Figure 1, the original strategy was to move as much of the Network NCP and File Package as possible out of the B4700, which is ill-equipped for such adventures, into a PDP-11/45. Thus, the PDP-11 was to act as a "front-end" machine for Network I/O (and incidentally was also to serve as a front-end in an entirely different sense: the user interface into NSW).

Thus, the NSW rje interface to the B4700 was to be split into the Network-protocol front end, and a relatively simple "back-end" with system calls and hooks within the B4700. The PDP-11 is connected to the B4700 through a local Binary Synchronous link, over which a simple command/response protocol is used; this latter was named "Interface Protocol", or "IP". Within the B4700 there is a system job that is a server for IP, simply translating the IP commands into system calls. As originally defined, therefore, IP was specific to the B4700 system.

Figure 1 : IP as Local Protocol Between PDP-11 and B4700

```
         T E N E X                          G U N T E R   A F S
        P D P - 1 0                    P D P - 1 1         B 4 7 0 0
xxxxxxxxxxxxxxxxxxxxxxxxxxx                             xxxxxxxxxxxxxxx
x                         x          xxxxxxxxxxxxxx     x             x
x                         x          x            x     x             x
x                         x          x            x     x             x
x :----------:            x          x            x     x :---------: x
x :          : :--------: x   <PCP>  x  :-------: x<IP> x :         : x
x :  WORKS   : :        : x Protocol x  :       : x     x :  I P    : x
x : MANAGER  : :  NSW   : <-------------:  NSW  : x     x :         : x
x : OPERATOR :-: Inter- :-------------->:inter- :<------->: SERVER  : x
x :          : :   host : x  ARPANET x  :  host : x     x :         : x
x :  (WMO)   : : Proto- : x          x  : Proto-: x L L x : Program : x
x :          : :    col : x          x  :   col : x o i x :         : x
x :          : :--------: x          x  :-------: x c n x :         : x
x :----------:            x          x            x a k x :---------: x
x                         x          x            x l   x             x
x                         x          xxxxxxxxxxxxxx     x             x
xxxxxxxxxxxxxxxxxxxxxxxxxxx                             xxxxxxxxxxxxxxx
```

Figure 2 : IP Extended Across ARPANET to WMO Host

```
         T E N E X                          G U N T E R   A F S
        P D P - 1 0                    P D P - 1 1          B 4 7 0 0
                                                        xxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxx          xxxxxxxxxxxxxx     x             x
x                         x          x            x     x             x
x                         x          x            x     x             x
x                         x          x            x     x             x
x :----------:            x          x            x     x :---------: x
x :          : :--------: x  <NETIP> x  :-------: x<IP> x :         : x
x :  WORKS   : :        : x          x  :       : x     x :  I P    : x
x : MANAGER  : :  I P   : <-------------:Network: x     x :         : x
x : OPERATOR :-:  USER  :-------------->:inter- :<------->: SERVER  : x
x :          : : PROCESS: x  ARPANET x  :  face : x     x :         : x
x :  (WMO)   : :        : x          x  :       : x     x :         : x
x :          : :--------: x          x  :-------: x     x :         : x
x :----------:            x          x            x     x :---------: x
x                         x          x            x     x             x
x                         x          x            x     x             x
xxxxxxxxxxxxxxxxxxxxxxxxxxx          xxxxxxxxxxxxxx     xxxxxxxxxxxxxxx
```

Figure 3 : Network IP Server Inplemented on 360/91

```
        T E N E X                           U C L A  C C N
       P D P - 1 0                            3 6 0 / 9 1
                                    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxx          x                                x
x                        x          x                                x
x                        x          x  "IPTASK"            "MP"      x
x                        x          x                                x
x :----------:           x          x                    :---------: x
x :          : :-------: x  <NETIP> x  :-------: <CCNIP> :         : x
x :   WORKS  : :       : x          x  :       :         :   I P   : x
x :  MANAGER : :  I P  : <-------------:Network:<--------:         : x
x : OPERATOR :-:  USER :-------------->:inter- :-------->: SERVER  : x
x :          : :PROCESS: x  ARPANET x  :  face : Exchange:         : x
x :  (WMO)   : :       : x          x  :       :         :         : x
x :          : :-------: x          x  :-------:         :         : x
x :----------:           x          x  NCP    .          :---------: x
x                        x          x         ..........>:   TCAM  : x
x                        x          x      (pseudo-user) :---------: x
xxxxxxxxxxxxxxxxxxxxxxxxxx          x                     TSO Job    x
                                    x                                x
                                    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

This initial approach ran into difficulties, because the PDP-11 hardware/operating system combination was unable to support the large amount of code and tables required. Furthermore, there were some delays in development of the necessary interhost protocols. As a result, the model shown in Figure 2 was adopted for NSW. The processing functions which could not be handled in the PDP-11 were moved into the PDP-10 (Tenex) system on which the prototype WMO was running. The IP protocol was thus "stretched" over the ARPANET from the PDP-11 to the PDP-10 (Ref. 4), reducing the PDP-11 to the role of a full-duplex IP pipeline between the B4700 and the ARPANET.

It was realized at CCN that the IP Network protocol used to implement Figure 2 could be used for any batch host. Analysis of IP revealed that, although it was dependent in some details on the B4700, it could be easily mapped into IBM's OS/360 operating system. While the file system of the B4700 is much simpler than that of OS/360, both systems contain the same concepts: logical and physical records, blocking factors, space allocation in discrete glumps, and ASA carriage control for print files.

These considerations led finally to the model of Figure 3, whose implementation on the 360/91 has provided an interim remote job entry capability for NSW. The 360/91 is a large machine with a complex multi-tasking operating system, and is easily capable of supporting both the Network interface (Network Control Program, or NCP; see Ref. 5), and the IP Server Message Processor, or "MP". The CCN-developed interprocess-communication facility, the "Exchange" (Ref 6), provides a convenient mechanism for passing data between MP and the NCP. The design of the CCN IP Server program is described in Section 3 below.

1.4. PROTOCOL DESIGN CONSIDERATIONS

Although the original definition of IP was basically suitable, some changes in the protocol were required. These changes were negotiated between Charles Muntz of MCA, who wrote the IP User program, and CCN. Some changes were trivial generalizations of B4700-specific protocol details. For example, file names had to be lengthened, to handle the 44-character names allowed in OS/360; job names had to be allowed in place of job numbers; and much larger blocking factors had to be allowed (typically 44 records-per-block on the 360, compared with 5 on the B4700). In addition, however, some significant protocol issues arose, mainly in the area of reliability and recovery from failures of user and/or server host. We will now discuss these protocol issues in IP. Section 2 below contains a complete definition of the resulting protocol.

1.4.1. NETWORK CONNECTIONS

As a result of its orientation towards a mini-host (the PDP-11), IP assumes that the Server process is "listening" on a fixed pair of sockets for the User process to make a connection. For simplicity this model was maintained, and a pair of sockets was chosen that doesn't conflict with ICP sockets or CCN socket allocations: 258 and 259. The Server is completely passive; he only listens for RFC's on these sockets. The IP User process (i.e. the WMO) must initiate the connections by sending RFC's. If the Server does not respond, the User process must time out, wait a reasonable interval, and try again.

Another question which arose was whether to keep the User-Server connections open "permanently" or to let the User close them when there are no jobs in progress. The latter choice was made, although it is not a very critical issue; the only expensive system resource tied up at CCN is the input buffer in the NCP.

1.4.2. TIME AND CHARGES

There was concern that IP provided only one "shot" at returning the time and charges for a batch job to the WM, namely an asynchronous (unsolicited) Job-Done message. There is a small but significant probability that this message will be lost due to User or Server crashing at the wrong moment. The WM must be able to keep an accurate accounting of job charges, and this requires that charges be reported to it in a reliable manner. To improve this reliability, MCA changed the WMO to tolerate multiple copies of the same Job-Done message, and the CCN Server was designed to resend the Job-Done message containing time and charges under the following conditions:

* when the WMO requests status for a job for which a Job-Done message has already been sent (presumably, WMO would not request status if he had already received a Job-Done message);

* when the Server receives a SYS Reset message and has batch output which the WMO has not yet retrieved.

These changes did not however solve the entire problem. For a long-running job, CCN takes 'check-point' accounting records; if the system crashes before the job terminates the user is actually charged for the cost accumulated at the last accounting checkpoint. If NSW is to support very large jobs, the WM must be kept informed of the accumulation of job charges. CCN takes an accounting checkpoint roughly every ten minutes of CPU time (actually every 1000 MUS, where MUS is the time-like machine resource-utilization measure at CCN). We changed the IP reply message to the Job-Status probe to include time and charges. The WMO can easily probe the status of jobs at CCN once a minute without undue overhead, and the reply keeps it informed of the accumulation of job charges.

1.4.3. RESYNCHRONIZATION

To bring the IP User and Server processes back into synchronism after a crash, an IP SYS Reset message was introduced. The WMO sends a SYS Reset message whenever it comes up, or whenever it reconnects to the Server after the latter has been down. Upon receiving a SYS Reset message, the Server reinitializes itself and then echoes the SYS Reset. The two ends should now be synchronized.

Since the User side may go down at any time, the Server may get a SYS Reset message, for example, while a data-transfer is in prgress. In this case, the Server must terminate the transfer as if he had gotten an FTS Delete Transfer message (which includes scratching any partially-written file at CCN) and then reinitializes itself.

After echoing the SYS Reset message, the Server checks for any NSW job output waiting for retrieval; if any exists, he resends the Job Done messages for these jobs, as discussed above.

1.4.4. JOBNAMES

It is necessary that WMO and the IP Server keep their active-job tables in agreement with each other. It is very easy to generate scenarios wherein crashes at one end or the other cause confusion about which jobs exist in the system, and about the IP names of these jobs. The WMO internally uses a number (table index) for keeping track of active jobs. As a result of a Submit Job message, the Server returns a (unique) 8-character name for the submitted job, and the WMO records this in its table for later use. The Server, on the other hand, may have its own tables of active NSW batch jobs. In turn, the IP Server has to keep its job table in synchronism with all batch job tables internal to the operating system.

After discussion between CCN and MCA, it was decided to circumvent the problem of job table synchronization in the following manner. The WMO internal job number was added to the Submit Job message, to be used by the IP Server at CCN to create unique job names for NSW jobs. The Server would not have a job table, thus (in theory) eliminating the problem of table synchronization. A batch job at CCN is entered in various OS/360 tables as it moves through the stages of batch execution, but as long as each job has a different name, no confusion can arise.

It was assumed that the WMO would have a reliable data base for its job table, so it would reliably furnish unique numbers. It should be noted that this approach will work badly if the WMO does not in fact have a reliable data base, and as a result reuses job numbers; when it erroneously sends a Submit-Job message specifying a job number which is still in use, the WMO will get back the Job Refused by TBH message from the Server. Recovery from this situation may be very difficult or at least awkward for the WMO. Further consideration of these issues is desirable.

1.4.5. JOB OUTPUT FILES

For the B4700, the WMO has to create the output file(s) for a job (to complete the JCL) before submitting the job. There was some concern about using this approach for the 360, whose operating system requires that disk space be (at least partially) reserved when the file was created. Therefore, considerable disk space could be tied up in the job output reservations while jobs awaited execution.

We therefore adopted a model which is more natural to the CCN system. A batch job writes its output into a large pool of transient disk space ("SYSDA") available to all jobs, and after termination a system process transcribes the output into a file with a standard name ("OUTPTT.<jobname>") in another pool of disk space. The WMO can retrieve this latter output file knowing the <jobname> which was returned at Submit-Job time. It should be noted, however, that the CCN implementation (see Appendix C) does include hooks necessary to follow the B4700 IP definition, should that turn out to be more appropriate.

The IP protocol as modified is fully defined in the following section as well as Appendices A and B.

2. NETWORK IP PROTOCOL DEFINITION

2.1. BASIC DESIGN OF IP

IP was originally described in documents distributed by Charles Muntz of MCA (see Ref. 4, Ref. 11). This section is adapted from those documents by appropriate editing. IP as described here is an Interim Protocol for communication between the NSW Works Manager Operator (WMO) and the IP message processor (MP) on a large-scale Batch Tool-Bearing Host (BTBH) such as the UCLA CCN system.

2.1.1. USER/SERVER RELATIONSHIP

MP is strictly a server. When sent a message it usually produces a single "response" message. WMO is strictly a user. It has no obligation to respond to MP messages, and normally expects a single response to each messge it transmits.

There are exceptions to the one-request/one-response convention. MP does not respond to each unit of file transfer that it receives. Likewise, all units of a single file transfer that MP passes to WMO are considered responses to a single message.

MP can also send "unsolicited" messages to WMO. Such messages arrive with a zero "handle" (see below) and are immediately recognizable by WMO. These messages become interrupts to WMO processes supervising the MP operations. Currently, three such messages are defined: Job Done, and two kinds of requests to terminate service.

2.1.2. MESSAGE FORMAT

In IP a "byte" is an arbitrary bit pattern, while a "character" is selected from a limited ASCII character set which is quite digestible by record-oriented systems like OS/360; it specifically excludes the format effectors CR, LF, TAB, FF, EOL, etc. Rather, (optional) format of ASCII records is specified as an ASA format code in column 1 of each record (see the discussions on file transfers for more detail).

All IP messages (both request- and response-type) have the same format: an 8-byte header followed by a variable length portion. Using a PL/I-like notation, an IP message looks like:

```
1 MESSAGE,
```

```
2 HEADER,
  3 SUBSYS CHAR(1),
  3 FUNCTION CHAR(1),
  3 HANDLE BIT(16),
  3 MODIFIER CHAR(2),
  3 DATALENGTH BIT(16),
2 VARIABLEPART CHAR(DATALENGTH)
```

Where:

* SUBSYS is a character as follows:

    A - File Transfer System (FTS)
    B - Catalog System (CAT)
    C - Work Order Executive (WOE)
    Z - System Messages (SYS)

* FUNCTION and MODIFIER are characters (3 in all) used to request particular functions of the subsystem.

* HANDLE associates messages with processes and is a two-byte integer.

* DATALENGTH is a two-byte integer that specifies the length of the variable part.

* VARIABLEPART is a string of characters or bytes, the length of which is determined by DATALENGTH. Its contents are determined by SUBSYS, FUNCTION, and MODIFIER.

2.1.3. HANDLES

The network connection between MP and WMO is viewed by them as a single pair of physical channels; however, MP consists of several asynchronous processes. WMO assigns a unique (mod 2**16) handle to each request sent to MP so that MP can process several requests concurrently. MP is expected to return the request handle in the HANDLE field of the corresponding response(s) so that WMO can establish the correspondence. Each message is assigned a handle one greater than the previous message. The handle of the IP message following a WMO cold start is 1, as is the handle of the message following one with a handle of 2**16-1. Zero is a reserved value; it identifies an unsolicited message.

2.1.4. MESSAGE DESCRIPTION NOTATION

For the purposes of detailing the format of described messages, we will use the notation:

* CAT(fn,modifier,variable part)

* FTS(fn,modifier,variable part)

* WOE(fn,modifier,variable part)

* SYS(fn,modifier,variable part)

where:

fn is a character,

modifier is two characters,

variable part is a BNF metavariable, described in Appendix A.

Unsolicited messages are indicated by an asterisk appended to the IP subsystem name; e.g. SYS*(3,00,) is an unsolicited message requesting service termination. All other messages have non-zero handles assigned as described earlier.

2.2. THE CATALOG SUBSYSTEM

The NSW local file catalog, as envisioned by IP, consists of a directory of unique file names. This directory is partitioned according to IP server ID, since it is intended to allow the possibility of a single TBH site serving more than one WMO. Thus there are two representations of a file name, and these may generally be used interchangeably:

* The simple name, which does not include the IP server ID, is a string of characters, organized as one or more index levels delimited by periods. Each index level consists of one to eight alphanumeric characters, the first of which must be alphabetic (in an IBM implementation "alphabetic" will include "@", "#", and "$"). The total simple name may not exceed 33 characters.

* The qualified name is formed by prefixing the simple name with the two index levels which constitute an IP server ID. It is then distinguished from a simple name by being enclosed in single quote marks, without internal blanks. The total qualified name may not exceed 46 characters, counting the quotes.

Conceptually, the directory associates three numeric file allocation attributes with each file:

* The Logical Record Size in Characters (LRSC);

* The number of Records Per Block (RPB);

* The Number of RECordS (NRECS) -- this represents the average number of records that the file is expected to hold, and has no bearing on the number of records actually contained in the file at a given moment.

Negotiation with the catalog system consists of reading or setting file names and attributes, never file contents. Currently, four basic functions are supported by CAT:

2.2.1. READ FILE NAME

CAT Read File Name tests the file directory to see if a particular file exists there. If so, it returns its allocation attributes. The formats of this message and its replies are:

```
Read File Name      = CAT(0,00,<file name>)

File Present        = CAT(0,00,<file spec>)
File Not Found      = CAT(0,82,<file name>)
MP I/O Error        = CAT(0,88,<file name>)
```

2.2.2. ENTER FILE NAME

CAT Enter File Name creates a new file on the server system, enters its name in the local directory, and returns the actual name, which need not have been fully specified by the user system. LRSC, RPB, and NRECS are supplied by WMO. The file is given an initial space allocation based on NRECS, but it is initially empty.

The file name requested may optionally contain up to seven occurrences of the "wild" character Question Mark. CAT is free to substitute any alphanumeric character in order to create a unique name, so wild characters should not be used as the first characters of any index level. The name actually used will be reflected in the normal File Entered response.

The formats of this message and its replies are:

Enter File Name	= CAT(1,00,<partial file spec>)
File Entered	= CAT(1,00,<file name>)
No Space	= CAT(1,84,<partial file spec>)
Duplicate Name	= CAT(1,86,<partial file spec>)
MP I/O Error	= CAT(0,88,<partial file spec>)

2.2.3. PURGE FILE

CAT Purge File deletes a file, removing its entry from the directory. The formats of this message and its replies are:

Purge File	= CAT(2,00,<file name>)
File Purged	= CAT(2,00,<file name>)
File Not Found	= CAT(2,82,<file name>)
MP I/O Error	= CAT(2,88,<file name>)

2.2.4. RENAME FILE

CAT Rename File renames the (existing) file specified in the first name to the (non-existent) second name. The formats of this message and its replies are:

Rename File	= CAT(3,00,<file name pair>)
File Renamed	= CAT(3,00,<file name pair>)
File Not Found	= CAT(3,82,<file name pair>)
Duplicate Name	= CAT(3,86,<file name pair>)
MP I/O Error	= CAT(3,88,<file name pair>)

2.3. THE FILE TRANSFER SUBSYSTEM

2.3.1. GENERAL CONCEPTS

There are two basic File Transfer operations: SEND and GET. We refer to the "passer" as the process which is sending records to the "receiver". During a SEND, WMO is the passer and MP is the receiver. During a GET the roles are reversed.

There are five transfer channels defined by FTS. Each channel may be engaged in a GET or SEND operation independently of the others. Thus up to five simultaneous file transfers may be in progress at any time.

Every file transfer in IP is done in two stages:

* Negotiation with the local file directory, using CAT. CAT never reads file contents but is solely concerned with reading and writing the directory.

* Transfer of file contents by a transfer of each logical record using FTS. It is intended that FTS change file contents only, and never the directory. However, two exceptions occur: the virtual datum NRECS may be changed when a file is overwritten; and a data set may be deleted in response to certain error conditions arising in FTS.

2.3.2. FILE FORMATS

IP deals only with sequential files consisting of 8-bit bytes organized into fixed-length records. The intent of FTS is to ensure that:

* Any supported TBH file can be copied by IP with enough information so that it could be restored - as in an archival system.

* Supported files can be shared among tools on different types of TBH's.

2.3.3. RECORD FORMATS

IP files are classified into three types. Within a file of a given type, all records obey certain formatting conventions;

* A-format (ASCII)

An A-format record contains 0 <= n <= LRSC ASCII characters. It is the passer's responsibility to remove trailing blanks, and the receiver's responsibility to restore them when appropriate. In any case, it is always MP's responsibility to translate between ASCII and EBCDIC at the appropriate times.

* F-format (Formatted ASCII)

  An F-format record is identical to an A-format record. However, the first character (expressed or implied) is an ASA format code. No other format effectors are permitted. If the TBH operating system supports this type of formatting, as will be the case in an IBM implementation, MP translates the first character exactly as it does the others.

* B-format (Binary)

  A B-format record contains precisely LRSC binary bytes. All 256 bit patterns are permitted, and no translation, truncation, or padding is performed by either system.

2.3.4. BASIC SEQUENCE OF SEND

Transfer of logical records of a file is an exception to the one-request/one-response rule of IP; receipt of data does not cause the receiver to return an IP message, but simply to ask for the next message(s). The actual sequence during a normal SEND operation is as follows, assumming required CAT operations have been performed:

* WMO requests a transfer channel for a send operation.

* MP responds with a channel number.

* WMO sends records; MP stores records.

* WMO sends an end-of-file indicator.

* MP closes the file and confirms a successful transfer.

* WMO receives the confirmation, completing the transfer.

Every message passed by WMO has a new handle, and MP responses use the handle of the last WMO message. Note that once the transfer has begun, WMO will not expect to hear from MP until after end-of-file is indicated. I/O errors or service interrupts will change this, however. Exceptional conditions on either system are handled by transmitting a Delete Transfer message to the other system. For WMO errors, this message is sent instead of the next data record or the end-of-file. For MP errors, it uses the handle of the last received WMO message.

This means that WMO must listen for responses even when none are required. After an abort, MP will simulate CAT Purge File on the output data set.

2.3.5. BASIC SEQUENCE OF GET

The sequence during a normal GET operation is as follows:

* WMO requests a transfer channel for a GET operation.

* MP responds with a channel number.

* MP sends records; WMO stores records.

* MP sends an end-of-file indicator and terminates the transfer.

* WMO receives the indicator and terminates the transfer.

Because only one WMO message is involved, only one handle is used. This handle is replicated in all messages passed by MP. Errors are handled similarly to those under SEND. MP must listen for possible Delete Transfer messages, and should one arrive, the new handle will be used by MP until the channel is successfully freed. Likewise, MP may pass a Delete Transfer message if a local error occurs.

2.3.6. RESYNCHRONIZING AFTER ERRORS

In order to keep MP and WMO synchronized accross all possible events during a file transfer, it has been convenient to define a finite-state machine modelling the behvior of an FTS transfer channel, once it is actively involved in a GET or SEND operation. The following model is condensed from MCA working papers (see Ref. 12). It applies to both WMO and MP. We assume the initial state is either PASSING or RECEIVING.

PASSING state:

```
     Local Error:        Close file,
                         Send Delete Transfer,
                         --> RESYNC state.
     Msg from Receiver:  Close file,
                         Send Transfer Deleted,
                         --> IDLE state.
     End of File:        Close file,
                         Send Normal EOF,
                         If WMO --> CLOSING state,
                         If MP  --> IDLE state.
     (otherwise):        Send data record.
```

RECEIVING state:

Local Error:	Delete file, Send Delete Transfer, --> RESYNC state.
Delete Transfer:	Delete File, Send Transfer Deleted, --> IDLE state.
Normal EOF:	Close File, Send Transfer Complete, --> IDLE state.
Data Record:	Write record and free buffer.

CLOSING state (exists only at WMO):

Transfer Complete:	--> IDLE state.
Delete Transfer:	Send Transfer Deleted, --> IDLE state.

RESYNC state:

Data Record:	Free buffer.
Normal EOF:	Free buffer.
Delete Transfer:	Send Transfer Deleted.
Transfer Deleted:	--> IDLE state.

IDLE state:

The channel is now free on both systems.

2.3.7. FTS MESSAGES

All file transfers are done with FTS IP messages, as follows:

2.3.7.1. START SEND FILE

FTS Start Send File initiates an IP SEND operation, specifying a file name and format mode. Normal response is the assignment of a transfer channel within MP. The referenced file name must exist, and have an assigned LRSC, RPB, and an allocation appropriate to the number of records in the new file. Any old file contents associated with the name are deleted. The formats of this message and its replies are:

Start Send File	= FTS(Ø,Ø1,<xfr spec>)
Ok to Send	= FTS(<xfr no>,ØØ,<xfr spec>)
No Free Channel	= FTS(Ø,81,<xfr spec>)
File not Found	= FTS(Ø,82,<xfr spec>)

2.3.7.2. SEND RECORD

FTS Send Record sends the next record of the specified file transfer. Normally it expects no response. However, MP is permitted to respond with Delete Transfer. The formats are:

```
Send  Record          = FTS(<xfr no>,00,<record>)

Delete Transfer       = FTS(<xfr no>,04,)
```

2.3.7.3. NORMAL EOF

WMO sends FTS Normal EOF instead of a data record to finalize a file transfer operation normally. MP responds with either Transfer Complete or Delete Transfer. The formats are:

```
Normal EOF            = FTS(<xfr no>,03,)

Transfer Complete     = FTS(<xfr no>,05,)
Delete Transfer       = FTS(<xfr no>,04,)
```

2.3.7.4. DELETE TRANSFER

WMO sends FTS Delete Transfer instead of a data record to abort a file transfer operation due to an error condition within WMO. MP responds with Transfer Deleted. The formats are:

```
Delete Transfer       = FTS(<xfr no>,04,)

Transfer Deleted      = FTS(<xfr no>,06,)
```

2.3.7.5. TRANSFER DELETED

WMO sends FTS Transfer Deleted as its last comment on an aborted transfer operation. MP does not respond in this case. The format is:

```
Transfer Deleted      = FTS(<xfr no>,06,)
```

2.3.7.6. START GET FILE

FTS Start Get File initiates an IP GET operation, specifying a file name, which must exist, and a format mode. The formats of this message and its initial replies are:

```
Start Get File        = FTS(0,02,<xfr spec>)

Ok to Get             = FTS(<xfr no>,00,<xfr spec>)
No Free Channel       = FTS(0,81,<xfr spec>)
File not Found        = FTS(0,82,<xfr spec>)
```

Following an OK to Get response, several (one for each record in the file) data transfer responses are made by MP:

Gotten Record = FTS(<xfr no>,00,<record>)

MP finally terminates the transfer by one of:

Normal EOF = FTS(0,01,<xfr spec>)

Delete Transfer = FTS(0,04,<xfr spec>)

If MP received Delete Transfer it responds with Transfer Deleted:

Transfer Deleted = FTS(<xfr no>,06,)

2.4. THE WORK ORDER EXECUTIVE SUBSYSTEM

Batch job execution under IP takes place in four stages:

1) Transmitting any input and command files not already at the batch TBH.

2) Submitting the command file to the TBH's local operating system for execution.

3) Retrieving result file(s).

4) Deleting the job from the TBH system.

Stages 1) and 3) are accomplished through CAT and FTS; stages 2) and 4) are performed via WOE (Work Order Executive).

2.4.1. SUBMIT JOB

WOE Submit Job names a card-image file which is to be turned over to the local operating system as the definition of a local batch job. The formats of this message and its responses are:

```
Submit Job          = WOE(1,00,<job spec>)

Submit Successful   = WOE(1,00,<job id>)
File Not Found      = WOE(1,82,<job spec>)
Job Refused by TBH  = WOE(1,88,<job spec>)
```

2.4.2. QUERY JOB STATUS

WOE Query Job Status is used to learn the current status of a job previously entered into the local system via WOE Submit Job. Before responding with its true reply, if the job status is found to be "done," this message will trigger retransmission of the unsolicited WOE Job Done message. The format of this message and its reply is:

```
Query Job Status    = WOE(3,00,<WMO job no>)

Status Reply        = WOE(3,<job status>,<job msg>)
```

Here <job msg> is a string suitable for human interpretation, and <job status> is one of the following:

00 - Job not found
01 - Job running
02 - Job in output queue
03 - Job in input queue
04 - Job waiting for reader
05 - Job done

2.4.3. JOB DONE

WOE Job Done is not a WMO message; it is rather an unsolicited message passed to WMO by MP, containing a job's time and charges. Because this is the only mechanism defined to relay charging information to WMO, and because WMO does not respond to the message in any way, careful measures have to be taken to ensure that the message is not lost. MP will transmit the message on the following occasions:

* When the local system signals MP that the job's status has just been changed to "done".

* Whenever WMO sends a WOE Query Job Status for a job, and the Job Done message has already been sent for that job. In this case, MP will retransmit the message BEFORE responding with Status Reply.

* Immediately after sending the response to a SYS Reset message (see below).

The format of this message is:

```
Job Done              = WOE*(4,ØØ,<job summary>)
```

2.4.4. DELETE JOB

WOE Delete Job causes the indicated job to be deleted from any tables kept by MP or the local Operating System. The job's status must be "done". The format of this message and its replies is:

```
Delete Job            = WOE(5,ØØ,<WMO job no>)

Job Deleted           = WOE(5,Ø6,<WMO job no>)
Not Done              = WOE(5,ØØ,<WMO job no>)
```

2.5. THE SYSTEM MESSAGE SUBSYSTEM

System messages are defined for the purpose of exchanging system status information. They contain no information relating to any particular file or any particular job. The following are currently defined:

2.5.1. ECHO

SYS Echo requests MP to respond by simply returning the request to WMO unaltered. No side effects are produced by this message. Its format is:

```
Echo                 = SYS(0,00,<msg>)
```

2.5.2. INVALID MESSAGE

SYS Invalid Message is sent by either system, when a message has arrived at that system garbled. Further processing is not clearly defined. The format is:

```
Invalid Message      = SYS(1,00,<msg>)
```

2.5.3. TERMINATION REQUESTS

MP can signal WMO that it wishes to terminate service by either of two unsolicited messages. SYS Service Termination means that MP intends to terminate gracefully, and the WMO should stop initiating file transfers; as soon as MP is idle, it will terminate itself. SYS Server Crash means that MP is terminating less than gracefully, and cannot wait to finish file transfers currently in progress. In either case, WMO must poll MP periodically until service is resumed. At that time, a restart sequence must be initiated; however, this is currently undefined. The formats of these messages are:

```
Server Crash         = SYS*(2,00,)
Service Termination = SYS*(3,00,)
```

2.5.4. RESET

SYS Reset is sent to MP by WMO to return MP to an idle state. MP simulates FTS Delete Transfer on all active file transfers, and discards any messages waiting to be processed. The RESET message is echoed to signal WMO that the two systems are again synchronized. After this echo, unsolicited WOE Job Done messages will be retransmitted for any jobs whose status is "done". The format of both the message and response is:

```
Reset                = SYS(4,ØØ,)
```

3. IMPLEMENTATION OF THE CCN IP SERVER

3.1. GENERAL STRATEGY

There were several possible strategies for an IP Server implementation at CCN. The IP Server could, like the CCN FTP Server, execute directly within the NCP. However, this would have required extensions to the operating system interface of the NCP, and would have adversely impacted the stability of the NCP during debugging of IP.

Other considerations in choosing a strategy were the desire for rapid implementation of this interim facility, and the certainty that the IP protocol would evolve during and after the original implementation. Therefore, it was desirable to write the Server in a higher-level language, which meant a penalty in core memory requirements.

A second possibility was to run the IP Server as a batch job (or equivalently, an OS/360 system task). This had the drawbacks that (1) the unmodified batch environment of OS/360 does not have all the required file system hooks, and (2) there would either be core storage and a protection key tied up at all times, or else there might be an appreciable delay (minutes) in getting a main storage region to start IP whenever WMO called.

Based upon these considerations, the following strategy was actually adopted for the IP Server implementation:

* MP is written largely as a PL/1 program, which calls assembly - language interface subroutines as necessary.

* MP operates as a command processor under TSO, using existing TSO commands as much as possible for manipulating the OS file system. Under TSO, MP is swapped out of core whenever it is idle.

* For debugging, MP can be executed from a user terminal and the services of TSO TEST are available. However, the production MP program is executed by a "pseudo-user". That is, an ever-present system process connects to TCAM (the terminal driver of TSO) as a virtual terminal, using Exchange, and sends a sequence of TSO commands to LOGON and execute MP.

* To perform Network I/O, MP opens a direct Exchange window to a process in the NCP called IPTASK. IPTASK manages the ARPANET connections to the WMO, and passes buffers back and forth between the Network and the Exchange window to MP. See Figure 3. IPTASK was also chosen to LOGON to TSO as the pseudo-user, since IPTASK must be up

whenever the IP Server is running.

3.2. SYSTEM INTERFACES

MP in turn required suitable interfaces into the CCN batch system. Specifically, the following interfaces were required (for more detail see Appendix C):

1) Job SUBMIT Interface

MP required a system call with the following parameters:

SUBMIT(<job ddname>, <charge number>, <password>, <jobname>,<output pathname>).

Here:

* <job ddname> is a handle on the file containing the job stream to be submitted;

* <charge number> is the account to which job execution is to be charged (we believe that the IP Server, not the WMO, should determine the account number for running NSW batch jobs);

* <password> is the batch password for <charge number> (which is therefore not required to be transmitted across the Network or known to WMO);

* <jobname> is the (unique) 8-character name to be assigned to the job;

* <output queue> is the designation of the OS/360 output queue to which NSW jobs are to be delivered;

* <output pathname> is a string which is passed through the job submission and job execution mechanism to the system process "SPOOL3", whose function is explained below.

The existing TSO SUBMIT command lacked many of these parameters, so a new system-wide submit interface was needed. An SVC was written which verifies the uniqueness of the <jobname>, saves some of the parameters in the CCN job table (TMT and TMD), and enqueues the submit request for the IBM Reader/Interpreter process called BRDR; see Ref. 7.

2) Output SPOOLing Process (SPOOL3)

In the CCN batch system, a single system process named "SPOOL3" is used to return the output of batch jobs to on-line systems -- URSA, TSO, and now MP. "SPOOL3" transcribes and rectifies the primary "printer" output stream (or "SYSOUT" in IBM parlance) from a batch job, creating a single data set on disk. It then signals the appropriate on-line system that the job output is available. To incorporate NSW jobs, SPOOL3 was changed to dequeue batch jobs from a new OS/360 queue ("W"), and to notify MP via the CCN system-wide message queueing facility called GMF (Generalized Message Facility). SPOOL3 was also changed to interpret the <output pathname> passed from the submit SVC; this string includes: (1) the GMF queue name to be used to notify MP of job completion; (2) the data set name to be assigned to the output file, and (3) the disk volume on which the output file is to be written.

3) Job Status SVC

In order to service WOE Query Job Status messages, a STATUS SVC was written, with a call of the form:

STATUS(<jobname> -> <status string>)

Included in <status string> are the input/output queue position if the job is awaiting execution or SPOOL3, or the stepname, accounting measures, and charges if the job is currently in execution. This string is returned to the WMO in WOE Status Reply messages.

4) Exchange Interface

An assembly-language interface to the Exchange interprocess communication SVC's was required (Ref. 8).

5) GMF Interface

Finally, an assembly - language interface to GMF was written. This interface includes a software interrupt thru which SPOOL3 signals MP of batch output availability. It also allows MP to read the accounting parameters (time and charges) and output pathname from the message placed in GMF by SPOOL3 (Ref. 9).

3.3. NCP INTERFACE PROCESS IPTASK

The process in the CCN NCP which interfaces between MP and the ARPANET is called IPTASK. It is expected that there will be more than one host running a WM and WMO, although at most one WMO is to be allowed per host. At present, there is a test version of the WM/WMO on the Tenex host BBNB, and a production version at host ISIC. Each WMO incorporates an IP User process, and there must be a corresponding process pair (IPTASK,MP) at CCN for each. Since the WMO does not do an honest Initial Connection Protocol sequence to open the Network connections, each IPTASK must know what host and socket-pair to use. The set of hosts running WM's is expected to be administratively determined and not rapidly changing, so that a table within the CCN NCP is used to determine the selection.

The CCN NCP appears to the operating system as a single task, but internally it uses a commutator to perform multiprogramming (as does MP; see the following section). We say that internally the NCP creates a set of "pseudo-tasks". The IPTASK logical processes are composed of a set of pseudo-tasks named IPTASKA, IPTASKB, and IPTASKC.

There is exactly one IPTASKA active in the NCP at all times. IPTASKA is the master controller , using a table to create an (IPTASKB, IPTASKC) pair for each IPTASK process, i.e. for each IP User host which CCN is to service. IPTASKA is created at the level of a host-control pseudo-task when the NCP is started. It waits one minute and then creates ("p-attaches") an IPTASKB pseudo-task for each host in its table.

An IPTASKB operates in the following manner:

* IPTASKB opens an Exchange window to TCAM to create a virtual TSO terminal. It then sends a TSO LOGON command over this connection, using a userid and charge number passed by IPTASKA from its table.

* When the virtual keyboard unlocks, indicating that LOGON is complete, IPTASKB issues the TSO command "NSWMP", which calls a command procedure to invoke MP as a processor.

* IPTASKB then creates its partner IPTASKC as a sub-pseudo-task.

* If the virtual keyboard ever unlocks again, indicating that MP has terminated and the TSO executive (TMP) is requesting a new command, IPTASKB repeats the two preceding steps (the previous IPTASKC will have terminated itself already; see below). However, IPTASKB

will repeat the sequence only a limited number of times: if MP terminates five times in a row, IPTASKB closes the virtual terminal window, ending the TSO session, and waits one minute before starting its sequence over from the beginning. This delay protects against excessive system utilization caused by a failing MP.

IPTASKC does the actual data transfer. It operates in the following manner:

* IPTASKC issues a Listen for the Network sockets (S+2,S+3) and a host whose number was passed from the table. Currently S is 256.

* IPTASKC issues an Exchange Open request to open an Exchange path directly to MP. Specifically, he executes:

```
EXOPEN MYTAG=IPTASK2,MYJOB=ARPA,
     YOURTAG=<TSO userid>,YOURJOB=*,CHANLIM=2
```

* When this Exchange window opens, IPTASKC waits (if necessary) for RFC's from the remote IP User. When the RFC's arrive, IPTASKC sends matching RFC's to fully open the Network connection.

* IPTASKC now enters a data transfer loop, waiting for output from MP on Channel 1 of the Exchange window, or input for MP on socket S+2. It simply retransmits such data to socket S+3 or to Exchange channel Ø, respectively.

* If one or more of the two Network connections and the Exchange window closes or has an error, IPTASKC simply terminates. This causes all open Network connections and the Exchange window to be closed by the NCP. The termination of MP in turn awakens IPTASKB, which then repeats the cycle.

3.4. CCN MESSAGE PROCESSOR MP

Although MP operates as a single task (i.e. process or fork) under OS/360, it is organized to support multiple concurrent IP operations. This internal multiprogramming of MP is achieved by means of a variant of the standard "commutator" or "coroutine" organization. That is, MP contains a set of asynchronous functions, each having its own locus of control and operating uninterruptibly with respect to the others. A particular MP function relinquishes control to another function only voluntarily, "blocking" itself until some event occurs.

Multiprogramming within MP is controlled by a program called PROCESS_DIRECTOR. An MP function blocks by returning control to PROCESS_DIRECTOR, which tnen scans its list of functions and enters the next next one which is unblocked. Since PROCESS_DIRECTOR scans the function list in a circular manner, this list is often called a "commutator", and the entry corresponding to a particular function (coroutine) is called a "slot". In the following, we will use the terms "asynchronous function", "function", and "slot" interchangeably.

The MP design differs from the normal commutator organization in that PROCESS_DIRECTOR does not save the location counter of a function when it blocks. Instead, each function must save its internal state (normally as an integer-valued switch) before blocking, and branch on this state switch when PROCESS_DIRECTOR reenters the function (always at its initial entry point). This design approach allows the support of PL/I routines under the commutator.

Synchronization of MP functions is achieved with standard OS/360 binary-semaphore-like objects called Event Control Blocks or ECB's. An ECB is a fullword in the user address space. Before relinquishing control to PROCESS_DIRECTOR, an MP function must store into its commutator slot the handle for an ECB. The function will become unblocked when this semaphore is signalled or "posted" by some other function (e.g. when a buffer has been placed on an input queue for the blocked function) or by an external event (e.g. an Exchange operation from IPTASK has completed). MP does not distinguish internal from external events. If it finds that all functions on the commutator are blocked, PROCESS_DIRECTOR issues a real OS/360 WAIT SVC for the entire list of ECB's, one per slot. This WAIT operation is issued via the PL/I-to-Exchange interface subroutine EXWAIT (Ref. 8).

The functions of MP generally communicate with each other by moving data buffers among internal queues. There are common (synchronous) subroutines for manipulating these queues. In most cases, functions are blocked waiting for a buffer to be placed in their input queue. The particular advantage of the coroutine organization is that, since the functions are uninterruptible, they can manipulate the queues without requiring mutual exclusion.

In summary, the major components of MP are:

* The Main Control module, which includes PROCESS_DIRECTOR;

* A number of "asynchronous functions", operating independently under control of PROCESS_DIRECTOR; and

* A variety of synchronous service routines callable by any of the asynchronous functions.

A general description of these components follows. For more details, see Appendix C.

3.4.1. THE COMMUTATOR

The Main Control module consists of the program 'MAINPGM' and its subroutine 'PROCESS_DIRECTOR'. When it is first entered, MAINPGM calls various initialization service routines and then calls PROCESS_DIRECTOR. The latter exits only when MP is terminating, in which case MAINPGM calls various cleanup service routines and itself exits to the TSO TMP (executive).

PROCESS_DIRECTOR supervises the execution of the asynchronous functions of MP, using a set of parallel vectors defining the commutator. A cross-section of these vectors forms a single slot of the commutator. These vectors are:

* ENTRYPTS -- a vector of ENTRY variables giving the initial entry points of the corresponding asynchronous function modules;

* RESETPTS -- a vector of entry variables giving the entry points for executing a RESET. When a module is entered at its RESET entry, it must terminate any operation in progress quickly but cleanly.

* ECBADS -- a vector of POINTERs to the current handles for the ECB's for which the functions are blocked. The format of these handles is defined by the parameter format of the EXWAIT interface subroutine.

* RETCODES -- a vector of fullwords to receive the EXWAIT return codes corresponding to ECBADS.

* SLOT_STATES -- a vector of halfwords in which the asynchronous functions maintain their current state codes. Main Control clears this vector to zero as a signal to each asynchronous function to perform local initialization, but does not reference it thereafter.

3.4.2. THE ASYNCHRONOUS FUNCTIONS

Asynchronous functions synchronize themselves via the vectors ECBADS and SLOT_STATES, using the following conventions:

* The function is written to wait on a single event at a time;

* The function keeps all state information that must be retained across a wait in safe data areas. All CSECTs of the function's code must be synchronously refreshable, i.e. must be overlayable at the time that control is returned to PROCESS_DIRECTOR.

* The function implements a wait (blocks itself) with the following sequence:

  1) Store a handle for the appropriate ECB into its slot in ECBADS; (If the event is the completion of a pending Exchange open or EXCHange request, then the PLOXI routine (Ref. 8) which initiated the request returned a handle for the completion event. For a non-Exchange event, the program should provide an aligned fullword as an ECB; the handle is the 2's complement of the address (POINTER) for this word.)

  2) Store the value 'Pending' into its slot of RETCODES;

  3) Store its state value into its slot of SLOT_STATES;

  4) Execute a RETURN ;

  5) Upon being re-entered by PROCESS_DIRECTOR, test the state code and reposition the location counter (i.e. branch) to the point where execution should resume.

* The function should make reasonable efforts not to cause real waits in any other way.

Each function may consider itself to be operating non-interruptibly with respect to the other functions of MP, except when it issues a RETURN as described above. For this reason, a routine which is executed by more than one function (slot) need not be fully reentrant.

The functions defined in MP are as follows:

* (five slots) FTS -- the File Transfer Service's five asynchronous data transfer functions.

  These functions all share the single routine FTS, which is capable of executing both IP SEND and IP GET operations.

* EXOUT -- the output Exchange handler.

  This function manages buffers queued for transmission over the outgoing channel of the Exchange window to the NCP and thence over the Network to the user process.

* EXIN -- the input Exchange handler.

  This function decodes the header of each incoming IP message and moves the buffer containing the message into the input queue for the appropriate function -- FTS, GENERAL, or STATUS.

* GENERAL -- the general IP Message processor.

  This function processes all incoming IP messages except those handled by FTS or STATUS. Thus, it processes the following IP messages:

    CAT Read File Name (read attributes of existing data set)

    CAT Enter File Name ("Allocate" a new data set)

    CAT Purge File (delete a file)

    CAT Rename File

    WOE Submit Job

    FTS Start Send File (initiate FTS SEND operation)

    FTS Start Get File (initiate FTS GET operation)

* STATUS -- processes Status Query and Job Delete messages.

  This function sends WOE Status Reply messages in response to WOE Query Job Status, processes WOE Delete Job, and retransmits the asynchronous Job Done messages for any jobs awaiting retrieval at the time of a SYS Reset operation. This function is separate from 'GENERAL' primarily because Query Job Status can sometimes return two response messages for one input message. This behavior does not fit well with GENERAL's design, which assumes one reply for each message. WOE Delete Job is included here because it shares some common logic with WOE Query Job Status.

* NOTIFY -- job completion notification.

  This function listens for notification from the CCN batch system that an NSW batch job is complete and its output is available for retrieval. NOTIFY passes this signal back to the WMO asynchronously, and then turns the job over to STATUS.

* OPR -- user interface to CCN operator.

  This function is currently null.

3.4.3. THE SYNCHRONOUS SERVICE ROUTINES

Synchronous service routines can be called from any asynchronous function without that function's being blocked. Some important synchronous service routines are:

* "ENQ" and "NDEQ" manage the buffer queues which are used to pass data between functions.

  ENQ accepts the address of a queue anchor and the address of a buffer, and hangs the buffer onto the queue. If any function was waiting for that queue, his ECB is posted.

  NDEQ attempts to obtain a buffer from a queue. If none is available, NDEQ executes part of the wait sequence (setting ECBADS and RETCODES), allowing the calling function to simply store his state value and RETURN to the PROCESS_DIRECTOR to wait for an available buffer. Parameters to NDEQ are the queue anchor address and the address of a 'WAITER' block. A WAITER block in turn contains the resulting buffer address (if the dequeue was successful), and the caller's slot number. NDEQ also returns a 1-bit flag which is false if dequeue was successful and true otherwise.

* "JOURNAL" creates a history log of MP activity, periodically spinning it off to a printer. JOURNAL accepts a character string and a message type code. JOURNAL contains logic to suppress output of certain message types based upon parameters for the run, so he can be called liberally, for every message that might ever be wanted.

* "COMMAND" accepts a TSO command string and a vector of known error codes that could result. The command is executed, the results analyzed, and a code returned indicating one of the states described in the vector

'BASIC_SITUATIONS'.

Other synchronous service routines provide irterfaces to the system calls for Exchange, for the Generalized Message Facility (GMF), and to the Job Submit function of the CCN routing SVC. See Appendix C for details.

4. CONCLUSIONS

The ultimate NSW plan is for every TBH to include a complex file transfer module called a "File Package" (see Ref. 3). The File Package, which will be capable of performing complex resource-allocation and conversion actions as well as simple data movement, will be used in place of all the file transfer primitives of IP (CAT and FTS). The File Package protocol in turn will be built upon a new interprocess - communication protocol called MSG. MSG will also be used by the WMO to request the job submit/delete/status functions of WOE. These latter will be implemented by a program running in an NSW environment established by the Foreman.

Designing and building MSG, the Foreman, and the File Package will be major software efforts. While these efforts are in progress, the interim IP server described in this report will allow the productive use of the CCN 360/91 as a batch TBH under NSW.

This document has described the CCN IP server as of April 1, 1976. It is recognized that this implementation is incomplete, and some further evolution is to be expected before the interim protocol IP is actually replaced by the next generation of NSW protocols.

4.1. EXTENSIONS

There are a number of capabilities which will be needed that are not now included in the present IP Server at CCN.

4.1.1. OPERATOR INTERFACE

The OPR function of MP is currently implemented as a null program. This means that the SYS messages Service Termination and Server Crash are not sent. This function should be implemented, and given a communcations channel to the CCN system operator. It may be that additional functions would be helpful here.

4.1.2. USER CONVENIENCES

It is felt that the CCN system could provide some helpful information to the NSW user, using WMO-controlled inquiry facilities. For example, during the early days of NSW it will be helpful for a user to be able to inquire about the current availability schedule for a particular tool.

4.1.3. MONITORING

Initial experience with the IP system indicates that further development will be expedited by some improved tools for monitoring the activities of the server programs. Because their TSO terminals are effectively managed by a systems program, the usual user windows to the outside world are not available.

4.1.4. JOB CANCEL CAPABILITY

The present definition of IP does not include any mechanism for cancelling a job submitted by WOE. We feel that this capability will eventually be needed.

4.2. PERFORMANCE IMPROVEMENTS

No comprehesive data have been gathered on the performance of the current IP server; however, casual observation indicates that single FTS transfers operate at between 1000 and 2000 baud. We do not consider this acceptable. Only further study will locate all areas requiring improvement, but some things are known to need work.

4.2.1. TEXT COMPRESSION

It has already been agreed with MCA that IP will be extended to include compressed data transmission before July 1, 1976. The technique used will be that already defined for future implementation in the NSW File Package (see Ref. 3). The FTS file type concept will be extended to include compressed-A, compressed-F, and compressed-B.

4.2.2. MESSAGE BLOCKING

The current IP server will be changed to pack and unpack IP messages explicitly, instead of using the implicit packing of the CCN Exchange service. This is expected to result in a lower swap rate for the TSO task that implements the server, by keeping control within that task for longer periods of time.

4.3. RELIABILITY IMPROVEMENTS

It has been noted that CCN intends to sophisticate the mechanisms used to keep the "job tables" at CCN and at WMO synchronized. This will be done by removing the the explicit relationship between WMO job number and CCN job name. IP will then have no serious problems recovering from a cold start on either end of the Network connections. A double cold start will still not be handled.

5. ACKNOWLEDGMENTS

The implementation of an IP Server at CCN was enormously aided by the constant cooperation of Charlie Muntz and his coworkers at Massachusetts Computer Associates, Inc. We appreciate Muntz's allowing us to freely plagiarize his IP documentation in order to prepare Section 2 and Appendix A of this document.

6. REFERENCES

1) "BASIC.ORIENTATION". CCN Document B001. Campus Computing Network, UCLA, Oct 31, 1975.

2) "Overview of NSW Batch", C. MUNTZ, Unpublished working document, MCA, Wakefield, Mass., 1975.

3) "File Package: The File Handling Facility for the NSW", Charles Muntz. Massachusetts Computer Associates, Inc., document CADD-7602-2011. MCA, Wakefield, Mass., Mar. 1976.

Also, "MSG Design Specifications". Chapter 3 in op.cit.

4) "Tenex IP Design", C. Muntz. Unpublished working paper. MCA, Wakefield, Mass., 1975.

5) "CCN ARPA Interface Software Program Logic Manual". CCN systems document Q43. Campus Computing Network, UCLA, Sept 1972.

6) "Programmer's Guide to the Exchange", R. T. Braden and S. Feigin. CCN Technical Report TR5. Campus Computing Network, UCLA, Mar 1972.

7) "Using the System Routing SVC to Submit Jobs", L. Rivas. CCN systems document S-179, Campus Computing Network, UCLA, Feb. 6, 1976.

8) "A PLI (Optimizer) Interface to Exchange", R. T. Braden. CCN systems document S-191. Campus Computing Network, UCLA, Feb. 20, 1976.

9) "PLGMF -- PLIX Interface to GMF", R. T. Braden. CCN systems document S-182, Campus Computing Network, UCLA, Oct 29, 1975.

Also, "Programming Using the Generalized Message Facility", D. Worth. CCN systems document S-180. Campus Computing Network, UCLA, Aug 25, 1975.

10) "The Guardian and Service SVC's". CCN systems document S-136. Campus Computing Network, UCLA, Feb 6, 1976 (latest revision).

11) "BASIC.RATES". CCN Document B002. Campus Computing Network, UCLA, Nov. 5, 1975.

12) "Tenex IP Handling of FTS Errors", P. Cashman. Unpublished working paper. MCA, Wakefield, Mass., 1975.

13) "The CCNDEVTP Macro", H. Ludlam. CCN systems document S-169, Campus Computing Network, UCLA, Jan. 30, 1975.

14) "The TMPMAC Service", P. Neilsen. CCN systems document S-176, Campus Computing Network, UCLA, May 12, 1975.

15) "SUPP.FORTRAN.SUBROUTINES". CCN Document S006. Campus Computing Network, UCLA, in revision at this writing.

16) "SUPP.TSO.PLI_ROUTINES". CCN Document S092. Campus Computing Network, UCLA, in revision at this writing.

7. APPENDIX A -- IP MESSAGE SUMMARY

The following list is organized into groups that correspond roughly to WMO messages. Each such group is headed by the WMO message, which is further designated by an asterisk (*) following its message name. This message is followed by the possible MP replies. This convention breaks down for the FTS messages, and where unsolicited messages are listed. In such cases, refer to the section "NETWORK IP PROTOCOL DEFINITION" for more complete explanations. This list is basically a condensation of that section. However, the BNF definitions listed here are definitive, and are referenced by the text section.

7.1. FTS MESSAGES

```
Start Send File *      = FTS(0,01,<xfr spec>)
Ok to Send             = FTS(<xfr no>,00,<xfr spec>)
No Free Channel        = FTS(0,81,<xfr spec>)
File not Found         = FTS(0,82,<xfr spec>)

Send  Record *         = FTS(<xfr no>,00,<record>)
Delete Transfer        = FTS(<xfr no>,04,)

Normal EOF *           = FTS(<xfr no>,03,)
Transfer Complete      = FTS(<xfr no>,05,)
Delete Transfer        = FTS(<xfr no>,04,)

Delete Transfer *      = FTS(<xfr no>,04,)
Transfer Deleted       = FTS(<xfr no>,06,)

Transfer Deleted *     = FTS(<xfr no>,06,)

Start Get File *       = FTS(0,02,<xfr spec>)
Ok to Get              = FTS(<xfr no>,00,<xfr spec>)
No Free Channel        = FTS(0,81,<xfr spec>)
File not Found         = FTS(0,82,<xfr spec>)
Gotten Record          = FTS(<xfr no>,00,<record>)
Normal EOF             = FTS(0,01,<xfr spec>)
Delete Transfer        = FTS(0,04,<xfr spec>)
```

7.2. CAT MESSAGES

```
Read File Name *     = CAT(0,00,<file name>)
File Present         = CAT(0,00,<file spec>)
File Not Found       = CAT(0,82,<file name>)
MP I/O Error         = CAT(0,88,<file name>)

Enter File Name *    = CAT(1,00,<partial file spec>)
File Entered         = CAT(1,00,<qualified file name>)
No space             = CAT(1,84,<partial file spec>)
Duplicate Name       = CAT(1,86,<partial file spec>)
MP I/O Error         = CAT(0,88,<partial file spec>)

Purge File *         = CAT(2,00,<file name>)
File Purged          = CAT(2,00,<file name>)
File Not Found       = CAT(2,82,<file name>)
MP I/O Error         = CAT(2,88,<file name>)

Rename File *        = CAT(3,00,<file name pair>)
File Renamed         = CAT(3,00,<file name pair>)
File Not Found       = CAT(3,82,<file name pair>)
Duplicate Name       = CAT(3,86,<file name pair>)
MP I/O Error         = CAT(3,88,<file name pair>)
```

7.3. WOE MESSAGES

```
Submit Job *          = WOE(1,00,<job spec>)
Submit Successful     = WOE(1,00,<job id>)
File Not Found        = WOE(1,82,<job spec>)
Job Refused by TBH    = WOE(1,88,<job spec>)

Query Job Status *    = WOE(3,00,<WMO job no>)
Status Reply          = WOE(3,<job status>,<job msg>)

Job Done              = WOE*(4,00,<job summary>)

Delete Job *          = WOE(5,00,<WMO job no>)
Job Deleted           = WOE(5,06,<WMO job no>)
Not Done              = WOE(5,00,<WMO job no>)
```

7.4. SYS MESSAGES

```
Echo *                = SYS(0,00,<msg>)
Echo                  = SYS(0,00,<msg>)

Invalid Message *     = SYS(1,00,<msg>)
Invalid Message       = SYS(1,00,<msg>)

Service Termination = SYS*(3,00,)
Server Crash          = SYS*(2,00,)

Reset *               = SYS(4,00,)
Reset                 = SYS(4,00,)
```

7.5. BNF DEFINITIONS

```
<file name>      ::= <simple file name>
                     |<qualified file name>
<simple file name> (1)
                 ::= <dsname>
<qualified file name> (2)
                 ::= '<dsname>'
<dsname>         ::= <label>|<label>.<dsname>
<label> (3)      ::= <upper letter>|<label><upper letter>
                                   |<label><digit>
<partial file name>
                 ::= <partial simple file name>
                    |<partial qualified file name>
<partial simple file name> (1)
                 ::= <partial dsname>
<partial qualified file name> (2)
                 ::= '<partial dsname>'
<partial dsname>
                 ::= <partial label>
                    |<partial label>.<partial dsname>
<partial label> (3)
                 ::= <upper letter>
                    |<partial label><upper letter>
                    |<partial label><digit>
                    |<partial label><wild character>
<alphanumeric>   ::= <letter>|<digit>
<digit>          ::= 0|1|...|8|9
<letter>         ::= <upper letter>|<lower letter>
<upper letter>   ::= A|B|...|Y|Z|#|@|$
<lower letter>   ::= a|b|...|y|z
<number>         ::= <digit>|<number><digit>
<wild character>
                 ::= question mark
<file name pair>::= <file name>;<file name>
<file spec>      ::= <file name>;<file map>
<partial file spec>
                 ::= <partial file name>;<file map>
<file map>       ::= <lrsc>,<rpb>,<nrecs>
<lrsc>(4)        ::= <number>
<rpb>(4)         ::= <number>
<nrecs>(4)       ::= <number>
<xfr no>         ::= 1|2|3|4|5
<xfr spec>       ::= <file name>/<fmt spec>
<fmt spec>       ::= A|F|B
<record>(5)      ::= <char record>|<byte record>
<char record>    ::= <IP char>|<char record><IP char>
<byte record>    ::= <byte>|<byte record><byte>
<byte>           ::= any 8-bit code
<IP char>(6)     ::= <alphanumeric>|<blank>|<punctuator>
<blank>          ::= a blank
<punctuator>     ::= "|%|&|'|(|)|*|+|,|-|.|/|#|$|@|
                       |:|;|<|=|>|[|¢|]|_|{|}|¬
                       |exclamation point|question mark
```

```
                      |vertical|grave||circumflex
<job spec>      ::= <file name>/<WMO job no>
<job id>        ::= <label>
<job status>    ::= 00|01|02|03|04|05
<job summary>   ::= <WMO job no>;<cpu time>,<charges>
<cpu time>(7)   ::= <number>
<charges>(8)    ::= <number>
<WMO job no>(9) ::= <number>
<job msg>       ::= <job id>:<msg>
<msg>           ::= <ip char>|<msg><ip char>
```

Notes:

* 1) Length limited to 33 characters.

* 2) Length limited to 44 characters.

* 3) Length limited to 8 characters.

* 4) Limit implementation dependent.

* 5) Length recorded in header.

* 6) All characters with ASCII codes (octal) 040 - 176.

* 7) Units Implementation dependent.

* 8) In cents.

* 9) 1 - 256.

8. APPENDIX B -- CCN IMPLEMENTATION DEPENDENCIES

CCN's IBM OS/MVT implementation of Network IP has certain restrictions and extensions.

8.1. DEFERRED MESSAGES

SYS Service Termination and SYS Server Crash are never sent. They will be implemented in a later version.

8.2. INVALID MESSAGES

When SYS Invalid Message is passed from MP to WMO, it will contain the handle from the bad message, and carry the entire bad message, including the bad header, in its variable data field. At this writing, however, it is not well defined which fields of the bad message have been translated from ASCII to EBCDIC, so WMO should probably not process anything but the handle.

If the message is received by MP, it will be treated as if it is itself an invalid message.

8.3. TIME REPORTING

In the unsolicited Job Done message, "cpu time" is in CCN Machine Unit Seconds (MUS). For more information on this measuring unit, see Ref. 11.

8.4. STANDARD SYSTEM OUTPUT

IP does not explicitly define the Standard System Output file to be used when a job is submitted. Each Batch TBH makes this definition. At UCLA it is written to a data set named

OUTPTT.<job id>

where "<job id>" is returned to WMO in the WOE Submit Successful response. In our implementation it is the 8-character MVT job name assigned by MP. This data set is allocated implicitly by CCN's output spooling system, so WMO must not allocate it. Likewise, it is purged implicitly by WOE Delete Job, although if WMO chooses to request an explicit purge prior to deleting the job, no one will object.

8.5. DUPLICATE JOB NUMBERS

IP does not yet treat the problems of resynchronization of the two systems in all error cases. CCN is in the process of developing schemes for handling some cases. However, in the meantime, if WMO attempts to submit a job with a WMO job number which is already present in CCN's system:

* If the old job's status is "done," MP will simulate WOE Delete Job and will then attempt to submit the new job.

* If the old job's status is not "done", MP will respond with "Job Refused by TBH". However, it will issue an MVT CANCEL command against the old job, under the assumption that it is illegitimate. This will not necessarily prevent an unsolicited Job-Done message from being sent, at some future time, for the old job.

8.6. RESET EXTENSION

For our convenience in debugging, we allow a SYS Reset message to carry a non-null text field, although the corresponding response will not. If the incoming text is the string 'END', MP will reset and terminate normally, without response.

8.7. JOB STATUS MESSAGES

The "<job msg>" component of the Status Response message will be more structured than is required by IP. The formats are as follows:

* JOB NOT FOUND (modifier = 05):

  The message is of the form:

  jobname: NOT FOUND

* JOB WAITING FOR READER (modifier = 04):

  The message is of the form:

  jobname: READING

* JOB IN INPUT QUEUE (modifier = 03):

  The message is of the form:

  jobname: INP c nnnxx OF mmm JOBS zzzzzz

  Where:

  "c" is the MVT input queue identifier, A-O. Normally, the lower-valued queues (A, B) contain express jobs and move quickly, while higher-valued queues (N, O) contain jobs requiring massive system resources, and move very slowly.

  "nnn" is the position of the job in the queue. Position 1 is next to be executed.

"xxx" is an appropriate suffix ("ST," "ND," "RD," "TH").

"mmm" is the number of jobs in the queue.

"zzzzzz", is only present if the job has been "held", in which case it is the literal string "(HELD)". A held job is one that will be scheduled for execution by operator command instead of MVT's automatic scheduling facilities. Setup jobs are an example.

* JOB RUNNING (modifier = Ø1):

The message is of one of two forms:

```
jobname: RUN ssss rrrK iiiiI ttttS ddd.cc
jobname: RUN In rrrK
```

Where:

"ssss" is the (possibly truncated) step name of the job step currently in execution.

"rrr" is the current region (in kilobytes) being used.

"iiii" is the cumulative I/O request count.

"tttt" is the cumulative CPU time used.

"ddd.cc" is the cumulative charges for the job, in dollars and cents. There is a floating "$" in this field, and if a WMO program were to wish to extract charges from this message, it would be well advised to locate the field by scanning for "$".

"n" is an MVT job initiator id. This form of the message is returned if the job is currently in inter-step scheduling. At such a time, the other values are not available in main storage.

* JOB IN OUTPUT QUEUE (modifier = Ø2):

The message is of the form:

```
jobname: OUT c nnnxx OF mmm JOBS zzzzz
```

Where:

"c" is the MVT output queue identifier, usually "W" for output to be returned to NSW.

"nnn" is the position of the job in the queue. Position 1 is next to be spooled.

"xxx" is an appropriate suffix ("ST," "ND," "RD," "TH").

"mmm" is the number of jobs in the queue.

"zzzzzz", is only present if the job has been cancelled, in which case it is the literal string "(CANC)".

* JOB DONE (modifier = Ø5):

The message is of the form:

jobname: READY FOR RETRIEVAL

8.8. IP FILE ALLOCATION ATTRIBUTES

MP does not keep explicit track of the LRSC, RPB, and NRECS attributes. CAT Enter File Name converts these into the similar MVT attributes LRECL, BLKSIZE, and SPACE. CAT Read File Name does an approximate reconversion; however, WMO must not expect complete accuracy except in LRSC.

The calculation of NRECS is strictly approximate, and is based on the OS LRECL, BLKSIZE, and SPACE attributes, as well as the characteristics of the device on which the file resides. For complete details, see the documentation of MP synchronous subroutine LOOKUP in Appendix C.

The limitations on LRSC, RPB, and NRECS imposed by CAT Enter File Name are actually those imposed by the physical device on which the data set is allocated, and the amount of free space currently available there. MP currently reserves enough primary space for NRECS records as described by LRSC and RPB, with enough potential growing room to accomodate approximately three times that many additional records. However, expansion of any MVT data set is always contingent on availabilibty of free space on the original disk volume (CAT does not now support multi-volume data sets). Currently, NSW data is stored on an IBM 2314 disk volume. This disk has a track length (which limits LRSC * RPB) of about 7294 bytes. Inter-block gaps, other than end-of-track, require about 15Ø bytes each. On such a disk, the preferred RPB for a card-image data set (LRSC = 8Ø) is 44.

8.9. FTS RECORD HANDLING

In FTS file transfers, certain potential error situations have been circumvented by the following conventions:

* A short binary record will be padded out to LRSC bytes with binary zero bytes.

* Any record of any format that is longer than LRSC will be truncated to length LRSC without comment.

* Format-F records will be written to disk in the MVT preferred printer-image record format -- VBA. This means that deleted trailing blanks are not restored. However, over-long records are still truncated. If formatted data _must_ be recorded in record format FB, they may be transmitted as format-A records. Since MVT does not record the transmission format, no more information will be lost. When FTS opens a data set as the recipient of a Format-F SEND operation, it forces the Record Format of the data set to VBA. If it was not already that, this change will imply a change in the value of the virtual datum NRECS. This value is estimated quite differently for fixed and variable records.

8.10. FILE NAMES

CAT Enter Name, when it is successful, will always return the form <qualified file name> (defined in Appendix A). This is source for the file names which it substitutes into the Job Control data to be submitted to the BTBH background system. In MVT, batch job definitions are interpreted outside the scope of implied qualification by the IP server ID. In all other cases where WMO sends a file name, it is for interpretation by MP itself, and WMO need not concern itself which form of file name it uses.

8.11. PDS MEMBERS

In the CCN implementation, individual members of Partitioned Data Sets (PDS's) have most of the attributes of sequential files, and can be manipulated by FTS; however, PDS's are not supported by CAT, so this feature is of limited usefulness.

9. APPENDIX C -- MP PROGRAM LOGIC

9.1. LOGIC OF MAIN PROGRAM

The main controller of MP is the routine MAINPGM, with its subroutines PROCESS_DIRECTOR and RESET. When it is first entered, MAINPGM calls various initialization service routines and then calls PROCESS_DIRECTOR. The latter exits only when MP is terminating, in which case MAINPGM calls various cleanup service routines and itself exits to the TSO TMP (executive).

PROCESS_DIRECTOR supervises the execution of the asynchronous functions (i.e. coroutines or processes) of MP. It uses a set of parallel vectors, a cross-section of which forms a single slot of a "commutator". PROCESS_DIRECTOR calls routine EXWAIT to wait on a set of events, one for each asynchronous function. EXWAIT returns when at least one function may proceed, having set event completion status for all functions. PROCESS_DIRECTOR calls each function for which it finds a completed event, and then calls EXWAIT again.

RESET is called by PROCESS_DIRECTOR whenever it finds switch RESET_REQD set (see function EXIN), and by MAINPGM at program initialization and finalization. RESET calls each asynchronous function, regardless of its event status, using the 'Reset' entry points instead of the working entry points used by PROCESS_DIRECTOR. At the conclusion of this loop, the MP system should be returned to an idle state, as required by the IP SYS Reset message.

9.2. LOGIC OF ASYNCHRONOUS ROUTINES

Each asynchronous function is associated with a routine, although some such routines serve more than one function (e.g., FTS). Each routine has two entry points, the addresses of which are recorded in the function's slot of vectors ENTRYPTS and RESETPTS. The ENTRYPTS entry is called by PROCESS_DIRECTOR whenever there is work to be done for the function. The RESETPTS entry is called by the same caller when it is necessary for the function to return itself to an idle state as quickly and cleanly as possible. Both types of entry are passed the slot number as a parameter.

9.2.1. THE EXIN FUNCTION

The EXIN function manages the input channel (#0) of the exchange window connecting MP to the network.

EXIN waits on available data from the Exchange window, and on free buffers to put them in. Whenever a buffer is obtained and filled, EXIN examines its System and Function codes and selects an internal work queue accordingly:

* If the message is SYS Reset, EXIN places the buffer on a special 1-entry queue, and sets switch RESET_REQD. If the text field of the message is "END", it clears switch RUNNING so that MP will terminate normally. EXIN exits at once so that MP reset can occur.

* If the message is WOE Query Job Status or WOE Delete Job, it is placed on the STATUS work queue.

* If the message is for FTS, and its function code is in the range 1-5, it is placed on the corresponding one of the five FTS work queues.

* In all other cases, the message is placed on the GENERAL work queue.

EXIN is on the receiving end of a Stream-to-Move-mode Exchange channel. The data on this channel consists of discrete messages with fixed length headers and variable-length text fields, although IPTASKC on the other side of the window is unaware of this. In order to avoid the problems of deblocking, EXIN currently uses two complete Exchanges to acquire each message. The first acquires the 8-byte header, which includes the length of the text field, and the second acquires the text itself.

EXIN'S Reset entry is essentially redundant, and does nothing important.

9.2.2. THE EXOUT FUNCTION

The *EXOUT function manages the* output channel (#1) of the Exchange window connecting MP to the Network.

EXOUT waits on output buffers enqueued for him by any other components, and schedules these buffers through routine EXCH. Whenever an Exchange operation is complete EXOUT hangs the corresponding buffer on the free queue.

EXOUT is on the source end of a Stream-to-Move-mode Exchange channel; however, it does not attempt to block data buffers per se. Rather, it attempts to maximize the depth of queueing of EXCH requests. For this purpose, it builds an NX-position pipeline, setting up to manage the number of buffers specified by the NX datum in the parameter deck (see MP EXECUTION PARAMETERS).

EXOUT normally waits on the output buffer queue, which is EXOUT's input work queue, and moves buffers thus acquired into the front end of the pipeline by passing them to the EXCH routine. Before giving up control, *EXOUT* always goes to the other end of the pipeline and pulls out all buffers marked completed by Exchange. These are hung back on the free queue. The only time that EXOUT releases control specifying an Exchange as his pending event is when there are buffers to stuff into the pipeline, the pipeline is full, and the last buffer in the pipeline is still not marked complete by Exchange.

When EXOUT is entered for the first time, with its processing state set to CLEAR, it allocates and initializes the pipeline, and changes to state GETOUT.

When EXOUT is entered for Reset, unless it has never been initialized, the following events occur:

* EXOUT's work queue is copied to the free queue.

* Any EXCHanges pending on EXOUT's Exchange channel are WAITed for (IPTASK on the other side of this window is not equipped to handle the RESET function of Exchange).

* Any buffers found in the pipeline are freed.

* The processing state is reset to CLEAR.

9.2.3. THE FTS FUNCTIONS

The five FTS functions all share a single routine, which, because of MP programming conventions, does not need to be reentrant. The purpose of this routine is to copy data records, either from the Network to a data set or vice versa.

FTS operates under the control of a processing state that varies among six values. These are discussed in the subsequent sections. The routines processing these states are all entries in a common internal procedure. This is so they can reference a scope of definition that does not treat the five FTS data structures as a vector, and yet avoid undue proliferations of data items. Entry to one of these routines is always on behalf of a specific FTS slot.

When FTS is entered for Reset, it copies its work queue to the free queue. If a file is currently open, it is closed and FTS issues the TSO command:

```
FREE F(FTSn)
```

If the interrupted operation was a SEND, FTS issues the command:

```
DELETE 'name'
```

and updates its history file (see synchronous subroutine DSNHIST) accordingly. The processing state is set to CLEAR.

9.2.3.1. THE CLEAR STATE

When entered in CLEAR state, FTS checks its history file (See synchronous subroutine DSNHIST) to see if this slot owns a data set that was not scratched, due to an abnormal termination. If so, FTS issues the TSO command:

```
DELETE name
```

Where "name" is found in the history file. The file is then updated. In any case, the state is changed to IDLE, and FTS waits on its work queue.

9.2.3.2. THE IDLE STATE

The IDLE state handles all events that occur when the slot is not involved in a file transfer. The events that drive this state are all incoming messages. Three cases are distinguished:

* Delete Transfer is echoed as Transfer Deleted.

* Transfer Deleted is ignored, and its buffer freed.

* Any other message is moved to the data portion of a SYS Invalid Message message and this is returned.

Exit from IDLE state occurs under the control of subroutine FSTART of function GENERAL.

9.2.3.3. THE START STATE

The START State is set by subroutine FSTART of function GENERAL whenever an FTS slot is assigned to a Start Get File or a Start Send File. At the same time, the request buffer is hung on the FTS slot's work queue. The only reason for making this state change is to avoid FSTART's assigning two such requests to the same slot; otherwise this processing could occur in state IDLE.

FTS breaks down the input message and sets the DSNAME, transmission direction and mode, request handle, and similar values in its data area. FTS issues the TSO command:

```
ALLOC F(FTSn) DA(name)
```

Where "n" is the FTS transfer channel number (1 - 5), and "name" is the result of applying OUTDSN to the name supplied by WMO. For SEND operations in "F" mode, routine MAKEF is called to convert the data set characteristics to those preferred for printer images (VBA 137). In any case, LOOKUP is called to determine the actual data set characteristics, and the file is OPENed.

If at any time during this process an error occurs, FTS issues the TSO command:

```
IFREE FTSn
```

and abandons the process. If no errors occur, then the processing state is set to either PASS or RECEIVE. If RECEIVE, the FTS history file (see synchronous subroutine DSNHIST) is updated so that the data set will be scratched on recovery from any crash.

In all cases, the selected reply is enqueued for EXOUT, and FTS waits, in its new mode, for an input buffer.

9.2.3.4. THE PASS STATE

The PASS state performs the GET operation. The event that drives this state is the availability of a free buffer. Because FTS does not attempt to control the implicit wait in its PL/I READ statement, it always returns to let the commutator go around, whether a free buffer is immediately available or not.

On being entered in PASS state, if FTS finds an incoming message for this transfer, it can only mean that WMO has aborted the transfer. FTS closes and frees the file, returns a Transfer Deleted, sets the processing state to IDLE, and waits for another input buffer.

If FTS finds a local error condition, it closes out the transfer similarly, but returns Delete Transfer and sets the processing mode to RESYNC.

If FTS finds an end-of-data condition it closes out the transfer similarly, but returns Normal EOF and sets the processing Mode to IDLE.

If FTS reads a record from the data set without encountering any exceptional condition, it returns the record and leaves the processing state *unchanged*. Trailing blanks are truncated from such records except in the case of binary data. Binary records are forced to be of length LRSC by either truncation or padding with zeros.

Whenever FTS exits PASS state for any reason, it issues the TSO command

```
FREE F(FTSn)
```

and writes a statistical record to the journal. This record includes the transfer channel number, the number of records transmitted, and the average rate of transmission.

9.2.3.5. THE RECEIVE STATE

The RECEIVE state performs the SEND operation. The event that drives this state is the arrival of an input buffer. Like the PASS state, RECEIVE does not attempt to control implicit waits in its PL/I WRITE statement. However, because the controlled wait for input data is expected to be the limiting factor here, RECEIVE does not give up control unconditionally just to allow the commutator to go around.

If FTS receives Delete Transfer it closes the file and issues the TSO command:

```
DELETE name
```

Where "name" was saved from the process of initializing the transfer. It then returns Transfer Deleted and enters IDLE state.

If FTS finds a local I/O error, it also deletes the data set, but returns Delete Transfer and enters RESYNC state.

If FTS receives Normal EOF it issues the TSO commands:

```
RELEASE name
FREE F(FTSn)
```

Where "n" is the FTS transfer channel number. It then returns Transfer Complete, and enters IDLE state.

If FTS receives a data record and no exceptional conditions arise, it writes the data to its output file, frees its buffer, and remains in RECEIVE state, waiting for the next input buffer. If the record format of the output data set is "F", the record is forced to that length by padding or truncation (padding is by blanks for character data, zeros for binary). Otherwise, the data is written as is unless truncation is necessary to avoid exceeding LRECL.

Whenever FTS exits RECEIVE state for any reason, it writes a statistical record to the journal. This record includes the transfer number, the number of records transmitted, and the average rate of transmission.

9.2.3.6. THE RESYNC STATE

The RESYNC state resynchronizes a slot with WMO after FTS has aborted a transfer. The event that drives this state is the arrival of an input buffer.

If FTS receives Delete Transfer it returns Transfer Deleted and remains in RESYNC state.

If FTS receives Transfer Deleted, it frees the input buffer and enters IDLE state. Resynchronization is now effected.

If FTS receives anything else, and the aborted transfer was a SEND, the buffer is simply freed. However, if the aborted transfer was a GET, the entire message is returned in the data field of a SYS Invalid Message

message. In neither case is the processing state changed.

9.2.4. THE NOTIFY FUNCTION

The NOTIFY function accepts no input from the Network, but generates unsolicited Job Done messages as output. NOTIFY's primary waits are on a "notify" ECB connected to the Generalized Message Facility (GMF) and on the free buffer queue. Even when GMF is found to be non-functional, NOTIFY waits on the GMF ECB. It will be POSTed periodically by the TIMER function, thus waking NOTIFY to attempt to reconnect to GMF.

During normal operation, NOTIFY sleeps until GMF POSTs its "notify" ECB. Then NOTIFY reads the GMF message queue selected for notification (see MP EXECUTION PARAMETERS), copying all messages to the jobs-in-output table using routine JTADD. Routine MAKENOT is called to format and send unsolicited Job Done messages, and the GMF messages are deleted from the notification queue. These operations are ordered to minimize the effects of interruptions caused by loss of contact with GMF.

If NOTIFY finds that the selected notification queue is nonexistent or has the attribute "temporary," it issues an ALARM call to JOURNAL, which has the effect of aborting MP.

On being called for Reset, unless the processing state is already CLEAR, Notify resets it to that and closes its GMF window.

9.2.5. THE STATUS FUNCTION

The STATUS function responds to WOE Query Job Status and WOE Delete Job messages, and maintains the jobs-in-output table. It waits for message buffers from EXIN, and, occasionally, for free buffers. The present version does not use the asynchronous features of the GMF interface package, so those waits are uncontrolled.

When STATUS is entered in state CLEAR, it uses JTREAD repeatedly, constructing the in-core masks of the jobs-in-output table.

When entered in state RESETTING, STATUS scans this table, using JTREAD and MAKENOT selectively to transmit unsolicited Job Done messages for all jobs in output. Since this involves the aquisition of free buffers, there is a companion state, RSWAITBUF to control possible buffer waits.

When entered in state PROCESS, STATUS will be holding either a Query Job Status message or a Delete Job message. For a Query Job Status, TMTSTAT is called to search the OS queues. If it is found, TMTSTAT's output is returned as the reply (see the section on TMTSTAT for the values returned). Otherwise, the jobs-in-output table is searched. If the job is found here, both an unsolicited Job Done message and a Status Reply message are formatted and sent. In this case, STATUS sets the modifier field to "05" and the data field to "jobname READY FOR RETRIEVAL". If the job is not found here, the "job not found" response originally built by TMTSTAT is sent. Since this process may involve the aquisition of free buffers, there is a companion state, GETBUF, to control possible buffer waits.

When entered in state PROCESS holding a Delete Job message, STATUS first locates the job. If it is in an OS queue, STATUS returns Not Done. Otherwise, JTDEL is called, and the Job Deleted response is returned. Notice that this does not mean that the job was actually "deleted," but that the job number, by one way or another, is free.

When entered for Reset, STATUS determines whether an actual Reset message has been received from the WMO, by testing switch RESET_REQD. If this switch is off, STATUS sets its state to CLEAR. If the switch is on, STATUS sets its state to RESETTING, and initializes the loop that will later be used to scan the jobs-in-output table. In either case, the STATUS work queue is freed.

9.2.6. THE OPR FUNCTION

The OPR function will monitor for, and respond to, local operator commands affecting MP execution. Because no such commands are currently defined, OPR is a null function.

When entered normally, OPR waits for work on special queue FREEZEANC. This queue is never POSTed, so OPR should never be reentered unless a Reset occurs. When entered for Reset, OPR returns without action.

9.2.7. THE GENERAL FUNCTION

The GENERAL function completes processing of all Network messages not specifically assigned elsewhere by EXIN. These messages share the property that they never return more than one reply buffer. GENERAL consists of a routine to select a processing function and a set of routines tailored to specific messages. These will be discussed below.

When called for Reset, GENERAL simply frees any buffers on its work queue.

9.2.7.1. THE FSTART PROCESSOR

FSTART gets all FTS messages that have not yet been assigned a transfer number - that is, FTS Start Send File and FTS Start Get File. The set of FTS slots is scanned to find one in state IDLE. If none is found, a No Free Channel message is returned. Otherwise, the request is re-enqueued on the selected FTS work queue, the processing state for that slot is changed to START (to prevent its re-use), and a switch is set so that GENERAL will not try to return an immediate reply for this message.

9.2.7.2. THE CREAD MESSAGE PROCESSOR

CREAD processes CAT Read File Name messages. It allocates file CREAD to the requested DSNAME and calls an Assembler-language routine, LOOKUP, to get the attributes of the data set thus allocated. If any error is found, an error reply is returned to GENERAL. Otherwise, LOOKUP's output is edited into the prescribed data field format and this is returned. CREAD uses routine OUTDSN to convert standard output DSNAME's to their actual form.

CREAD issues the following TSO commands:

```
IFREE CREAD
ALLOC F(CREAD) DA(name) SHR
FREE F(CREAD)
```

Where "name" is the result of applying OUTDSN to the name given by WMO.

9.2.7.3. THE CENTER MESSAGE PROCESSOR

CENTER processes CAT Enter File Name messages. First it breaks out the request's subfields, and may return an immediate error reply. It then builds TSO ATTRIB and ALLOC commands and executes them, using a fixed DSNAME. If this is successful, CENTER calls its private subroutine FINDNAME, which issues RENAME commands until either one is successful or the combinations allowed by the pattern of wild characters in the request are exhausted. This process uses Assembler-language routine PERM for wild character substitution.

If a RENAME succeeds, the DSNAME is returned to WMO. Otherwise, the dataset is scratched with a TSO DELETE command, and an error reply is returned. The TSO command sequence is typically:

```
FREE ATT(@@)
DELETE @.inits
ATTRIB @@ REC(F B) BL(blo) LRE(lre)
ALLOC DA(@.inits) F(CENTER) BLO(blo) SPACE(pri,sec)-
     USING(@@) VOL(vol) NEW
RENAME @.inits newname
FREE F(CENTER)
```

Where:

"inits" is the TSO Userid of the pseudouser driving this IP server session.

"blo" is LRECL * IP RPB * BF, where 'BF' is an input parameter (see MP EXECUTION PARAMETERS).

"lre" is IP LRSC.

"pri" is = IP NRECS / (IP RPB * BF), rounded up.

"sec" is "pri" / 5, rounded up.

"vol" is the AV input parameter (see MP EXECUTION PARAMETERS).

"newname" is a name generated by applying PERM to the skeleton name provided by WMO.

The RENAME command may be repeated. If any error is encountered, the above sequence is abandoned and CENTER issues the command:

```
DELETE @.inits
```

9.2.7.4. THE CPURGE MESSAGE PROCESSOR

CPURGE processes CAT Purge File messages. It issues the TSO command:

```
DELETE name
```

Where "name" is the result of applying OUTDSN to the data set name. CPURGE returns a result selected from the result of the command, without actually testing for success.

9.2.7.5. THE CRENAME MESSAGE PROCESSOR

CRENAME processes CAT Rename File messages. It issues the TSO command:

RENAME name1 name2

Where:

"name1" is the the result of applying OUTDSN to the old data set name. CRENAME uses OUTDSN for system consistency, even though it might be considered illegal to rename a Standard System Output data set.

"name2" is the new data set name.

9.2.7.6. THE SUBMIT MESSAGE PROCESSOR

SUBMIT processes WOE Submit Job messages. It breaks out its data field, allocates the data set to a standard file, calls OENSUB, frees the file, and returns the result of OENSUB. The charge number, destination class, password, and notify queue that SUBMIT passes to OENSUB are those supplied by the MP parameter data set (see MP EXECUTION PARAMETERS). SUBMIT uses OUTDSN for system consistency, even though it might be considered illegal to submit a Standard System Output data set.

9.2.7.7. PROCESSING OTHER MESSAGES

If the input message is SYS Echo, GENERAL does not process it at all, but simply enqueues the buffer for output transmission. If it is anything else, the entire message is placed in the data field of a SYS Invalid Message message, and that is returned.

9.2.8. THE TIMER FUNCTION

The TIMER function provides a basic "checkpoint" facility for MP. It maintains an outstanding timer interval of the number of minutes specified by parameter "TI" (see MP EXECUTION PARAMETERS), and on expiration of that interval, performs certain periodic functions.

9.2.8.1. JOURNAL PROTECTION

One periodic function attempts to minimize the risk that the MP journal will be lost due to system failure. For this purpose, whenever a checkpoint interval passes during which there has been no journal activity other than that explicitly requested from TIMER, then if the journal contains data other than TIMER's checkpoint records, then JDSFIN and JDSINIT are called, thus scheduling the current journal data set for printing.

9.2.8.2. STATISTICAL MONITORING

Another periodic function is the writing to the journal of statistical records noting the current level of MP's utilization of the Host Operating System's resources. TIMER uses subroutine ACTSTAT to acquire these statistics, and calls JOURNAL to write them.

9.2.8.3. GMF RETRY

TIMER posts NOTIFY's GMF notification ECB. If GMF has crashed and recovered, this should cause NOTIFY to reconnect. If all is well already, nothing is hurt.

9.2.9. SUMMARY OF ASYNCHRONOUS DEPENDENCIES

The asynchronous facilities of MP are not completely generalized. Many WAIT situations occur, in the processing of most messages, which will momentarily block the entire MP package. However, there is a level of synchronization which is under MP's control. This effectively divides IP messages into several groups. Within each group, all messages are processed absolutely sequentially. Between groups, MP will do what it can to overlap processing. These groups are:

9.2.9.1. RESET

When a SYS Reset message is received, it is acted on as soon as possible. However, this may not be until routines processing one of the other groups come to a WAIT situation.

9.2.9.2. NOTIFICATION

The unsolicited Job Done message, when it is NOT a retransmission, is asynchronous to all other messages.

9.2.9.3. STATUS

The following messages are processed absolutely sequentially:

SYS Query Job Status
SYS Delete Job
Any <u>retransmitted</u> unsolicited Job Done message.

9.2.9.4. GENERAL

The following messages are processed absolutely sequentially:

FTS Start Send File
FTS Start Get File
CAT Read File Name
CAT Enter File Name
CAT Purge File
CAT Rename File
WOE Submit Job
SYS Echo
SYS Invalid Message
All messages with unknown Subsystem/Modifier codes.

FTS Start Send File and FTS Start Get File are special cases. Their processing is begun in this group. However, if a free FTS transfer channel is found, the rest of their processing is turned over to the FTS group. Between these two parts of processing, any other messages for the GENERAL group will be processed.

9.2.9.5. THE FIVE FTS GROUPS

Once FTS Start Send File or FTS Start Get File is turned over to FTS, all further processing of that message and of subsequent messages for that transfer channel are processed absolutely sequentially. There are five FTS channels, and each of these functions absolutely independently of and asychronously to all others.

9.3. LOGIC OF SYNCHRONOUS ROUTINES

The synchronous service routines can be called by any function at any time, without that function's risking losing control to PROCESS_DIRECTOR.

9.3.1. EXCHANGE MANAGEMENT PACKAGE

The Exchange Management Package is used to interface MP with IPTASK and hence the ARPA Network. It consists of the CCN PLOXI package and module EXDONE.

9.3.1.1. PLOXI PACKAGE

Modules of the PLOXI package that are used by MP are:

* EXCH

* EXOPEN

* EXCLOSE

* EXWAIT

These modules are fully documented elsewhere (see Ref. 8) and will not be further covered here.

9.3.1.2. EXDONE ROUTINE

EXDONE is a simple routine used by EXIN and EXOUT to set up the standard values needed to WAIT on a PLOXI ECB. EXDONE accepts an EXCH ID, an EXCH Return code, and a slot number, and returns a Boolean (1 bit) flag. If this flag is True then the Exchange is complete, and the caller must not wait. If it is False, the caller may wait by merely issuing RETURN.

9.3.2. JOB MANAGEMENT PACKAGE

The Job Management Package keeps track of the jobs created by MP as they move through the host operating system and into MP's GMF message queues.

9.3.2.1. GMF INTERFACE

MP uses the PLIX-callable interface routines to GMF (the Generalized Message Facility). In particular, the following modules from this interface package are used by MP:

* GMOPEN

* GMCLOSE

* GMUSER

* GMQUEUE

These modules are fully documented elsewhere (see Ref. 9) and will not be further covered here.

9.3.2.2. THE JOBMGT ROUTINE

JOBMGT is a group of entries used by both STATUS and NOTIFY to manipulate the jobs-in-output table. This table consists of a disk data set containing 256 138-byte GMF messages, initially set to all blanks. The table is summarized in core by two 256-bit masks: JOB_IN_OUTPUT identifies those job numbers for which there are non-null records in the data set, and STRANGE_DSN identifies those whose Standard System Output files bear other than the default name.

Routines provided are:

* JTREAD returns the file record for a job. In the process, it corrects the in-core masks, so it can be used for initialization.

* JTADD adds a job to the table. If it overlays an existing job, the old one is deleted, and an operator message is written.

* JTDEL deletes a job from the table. If the associated output data set still exists, JTDEL issues the TSO command

  DELETE 'name'

  where "name" is read from the jobs-in-output record.

* MAKENOT formats and enques a buffer for the unsolicited Job-Done message. MAKENOT accepts a pointer to an MP buffer and a pointer to a GMF message record (including the 20-byte time/date stamp area). On return, the buffer has been enqueued for EXOUT, so the caller no longer owns it.

* OUTDSN examines a DSNAME. If it is the name for a MP Standard System Output data set, and if that job has used another DSNAME, then OUTDSN returns the DSNAME actually used. Otherwise, it returns its input string. OUTDSN both accepts and returns CHAR(46) VAR. The input DSNAME can be in either of the two TSO forms, that is, either qualified and quoted, or simple and unquoted. The returned result is not

necessarily in the same format as the input.

9.3.2.3. TMTSTAT AND INTMT

TMTSTAT is an Assembler-language routine which formats a status reply for jobs still in the host operating system. It accepts a pointer to an MP buffer, the data area of which contains the job name to be queried, and returns a Boolean (1-bit) flag. This flag will be True if the requested job was found, otherwise False. In either case, a descriptive message will have been added after the 8-character job name in the buffer, and the MODIFIER field set according to this convention:

```
00 --> Job not found
01 --> Job running
02 --> Job in Output Queue
03 --> Job in Input Queue
04 --> Job reading
```

INTMT is an alternate entry which merely returns the Boolean flag.

This routine uses CCN Service SVC 11 (see Ref. 10) to locate the job step TCB address.

9.3.2.4. OENSUB ROUTINE

OENSUB is an Assembler-language interface to the SUBMIT function of the CCN Routing SVC (see Ref. 7). OENSUB returns a halfword result code and is called with the following parameters:

* The 8-character job name to be assigned the job (the name on the JOB card is ignored). By CCN convention, the job's Charge Number is the first six characters of the job name; the last two characters are derived from the <WMO job no>.

* The 8-character file name allocated to the data set containing the job.

* The 8-character destination name. (This is normally a SYSOUT class name.)

* The 8-character batch password for the job's Charge Number.

* The 60-character output path definition. This consists of the 44-character DSNAME to be given the MP Standard System Output (the data set which will receive the linearized output from the job), followed by the 8-character GMP Queue name to be used to notify MP of the job's completion, followed by the

8-character Volume Name of the Volume where the output data is to be placed.

OENSUB returns the return code from the Routing SVC. All the caller needs to know about this is that if it is nonzero an error occurred and the job was not submitted.

9.3.3. DATA SET MANAGEMENT PACKAGE

The data Set Management Package performs auxiliary functions for the data set manipulative parts of MP.

9.3.3.1. DSNHIST ROUTINE

The DSNHIST routine manages file DSNFILE. This file must be allocated to a data set containing five 56-character records, initialized to blanks. FTS will note there the names of data sets that should be scratched during crash recovery.

Entry HISTIN accepts a halfword transfer number in the range 1-5 and a varying string. The record corresponding to the number is read into the string.

Entry HISTOUT accepts the same parameters as HISTIN. The indicated record is overwritten by the string.

9.3.3.2. LOOKUP ROUTINE

LOOKUP finds and/or calculates the attributes and values associated with an allocated data set. If it encounters any error, it simply returns zero values. LOOKUP accepts:

* the 8-character file name allocated to the data set.

* the halfword fudge factor used to convert OS blocking factors into IP RPB. Normally, this is the BF parameter (see MP EXECUTION PARAMETERS).

* the address of an area of six halfwords, into which LOOKUP is to place:

  1) the OS Record Format ("U", "F", or "V"). The second character is set to blank.

  2) the OS LRECL.

  3) the OS BLKSIZE.

  4) the IP LRSC -- this is equal to LRECL after any count bytes have been subtracted.

5) the IP RPB -- if the data set has carriage control, this is set to 5. Otherwise, it is (MIN(MAXBLK,BLKSIZE/BF))/LRSC, where MAXBLK is currently 1000, BLKSIZE is used less 4 for variable records, and BF is an input parameter (see MP EXECUTION PARAMETERS).

6) the IP NRECS -- this is a rough approximation, calculated as follows:

* BLKSIZE is converted to Blocks Per Track (BPT) using the BTON service of the CCNDEVTP SVC (See Ref. 13).

* The number of blocks (NB) is the number of tracks in the data set times BPT.

* Unless RECFM is U, this is multiplied by RPB.

* Unless RECFM is F, this is multiplied by 3, under the assumption that the average record is probably 1/3 LRECL.

* The result is called NRECS.

9.3.3.3. MAKEF AND MAKEE

MAKEF is an Assembler-language routine which alters the characteristics of a data set which is about to be overwritten with printer image records. It accepts the 8-character file name allocated to the data set.

MAKEF forces the record format to "VBA". If it was not already "V", then MAKEF adds 4 to the LRECL and ensures that the BLKSIZE is at least 4 greater than that. It does all this in a DCB exit entered while OPENing a file to the data set.

MAKEE is an alternate entry to MAKEF. Its action is similar; however, instead of operating on the data set attributes, MAKEE makes the data set empty.

9.3.4. BUFFER MANAGEMENT PACKAGE

The Buffer Management package effects the passing of MP's internal buffers among various work queues, and the "POSTing" of functions waiting on those queues. For convenience and simplicity, some code translation and journalling services are invoked from within this package.

9.3.4.1. NDEQ ROUTINE

Routine NDEQ dequeues a buffer from a work queue for the caller. If there is nothing on the queue, the caller is added to the queue's wait chain and control values are set up such that the caller can wait for work merely by issuing RETURN.

NDEQ returns a Boolean (1-bit) result: False if a buffer was obtained and True if the caller should wait. This polarity is chosen to facilitate use of the form:

```
IF NDEQ ( . . . ) THEN RETURN;
```

NDEQ accepts a pointer to the work queue and a pointer to the caller's 3-word wait queue element. The first word of this element is the buffer pointer. The third word must have been initialized to the caller's slot number.

9.3.4.2. ENQ ROUTINE

Routine ENQ adds a buffer to a work queue, or, if someone is waiting on the queue, gives him the buffer and makes him dispatchable.

ENQ accepts a pointer to a work queue, a buffer pointer, and a halfword type indicator. The type code is used to control journalling and code translation. Set it negative to disable these services. If it is not negative, it should be selected from the array JOURNAL_MSGS defined in %INCLUDE block COMMON. JOURNAL will then be called, specifying the buffer and the type.

If the type is one of the "outgoing" types, and if translation is enabled, then the System, Function, and Modifier fields of the buffer are translated from EBCDIC to ASCII, and, unless the type is OUTGOING_BINARY, the Data portion is also translated.

If the type is INCOMING_DATA or INCOMING_MSG, and if translation is enabled, then the data field (only) of the buffer is translated from ASCII to EBCDIC.

9.3.4.3. COPYQ ROUTINE

Routine COPYQ accepts a pointer to a work queue. If buffers are enqueued in the queue, they are all dequeued and placed on the free queue. During this process, any waiters on the free queue will be dequeued without POSTing.

COPYQ is intended for use by the Reset entries of the asynchronous functions.

9.3.5. COMMAND MANAGEMENT PACKAGE

The TSO Command Management Package enables MP to invoke TSO command processors and to retrieve a status reply encoding the information that is normally written irretrievably to the TSO terminal. It consists of several parts:

9.3.5.1. COMMAND ROUTINE

The COMMAND routine accepts a character string to be interpreted as a TSO command, and a Pointer to a vector of error codes. It returns a halfword result indicating the most severe error that was found. The format of the error vector need not concern most callers, as they will use one of the following data from %INCLUDE packet COMMON:

ALLOC_ERRS
FREE_ERRS
ATTRIB_ERRS
DELETE_ERRS
RENAME_ERRS
NULL_ERRS

COMMAND returns a value selected from array BASIC_SITUATION declared in the %INCLUDE packet COMMON, namely:

OK
DUPLICATE_THING
THING_NOT_FOUND
IO_ERROR
TABLE_OVERFLOW
OTHER_ABORT

9.3.5.2. NSWCMD ROUTINE

Routine NSWCMD is an assembler-language adjunct to COMMAND. It has three entries:

9.3.5.2.1. COMEXEC is called by COMMAND, and it, in turn, calls a TSO Command Processor using the CCN TMPMAC service (see Ref. 14). COMEXEC's first parameter is the command string, and the second is a varying string to receive the concatenation of 6-character strings, each of which is a 5-character TSO message ID number followed by a blank. The length of this string indicates how many such codes are returned (note that the final trailing blank is truncated). COMEXEC returns a halfword value which is the code returned in GR15 by the TSO Command Processor.

9.3.5.2.2. COMINIT must be called before any other use is made of this package. It LOADs modules IKJPTGT, IKJGETL, and IKJPUTL from file IKJPUTL (this ensures that I/O using those names can be intercepted).

9.3.5.2.3. COMFIN should be called when this package is no longer needed to DELETE the modules LOADed by COMINIT.

9.3.5.3. IKJPUTL ROUTINE

Module IKJPUTL, alias IKJPTGT and IKJGETL, is an Assembler-language routine which is LOADed by COMINIT. Until it is deleted, its entries will intercept the calls generated by the Macros PUTLINE, GETLINE, and PUTGET whenever they are executed within the scope of its load-list entries. The actions of these entry points are:

9.3.5.3.1. IKJGETL: If COMMAND is currently active, control is returned to the caller with the "Attention" return code. Otherwise the true IKJGETL is called.

9.3.5.3.2. IKJPUTL: If COMMAND is currently active, and if an Informational Message is being passed for other than the Format-Only Function, and if the first segment (only) of this message is long enough to have a message identifier field, and if the fourth through eighth bytes are all numeric, and if there is still some room left in COMEXEC's error string, then the fourth through eighth bytes of the first segment of the message are appended to the string, followed, if there is room, by a blank. In any case, control passes on to the true IKJPUTL module.

9.3.5.3.3. IKJPTGT: If COMMAND is currently active, and if input is actually being requested, control is returned to the caller with the "Attention" return code. Otherwise, the true IKJPTGT is called.

9.3.5.4. ERRIDS SECTION

ERRIDS is a STATIC EXTERNAL section in which occur the standard error codes referenced by the %INCLUDE packet COMMON. It is generated during compilation of dummy routine COMMONS, and connected to the standard error code pointers during execution of routine GETPARM.

9.3.6. JOURNAL MANAGEMENT PACKAGE

The Journal Management package maintains a cumulative journal of all MP activities to which it is made privy. Various types of output records are defined by array JOURNAL_MSGS in %INCLUDE packet COMMON. Output of each can be disabled parametrically for each execution of MP (see MP EXECUTION PARAMETERS).

It is important that JOURNAL be called as often and by as many other routines as possible. Otherwise, the invoker of MP loses flexibility in tracking down problems.

The journal is normally printed, not kept for analysis. It is scheduled for printing every time a specified number of lines is written (see MP EXECUTION PARAMETERS), or whenever an external routine (such as the TIMER function) calls the appropriate entries.

Journal data can be independently routed to three different destinations: a SYSOUT class, the TSO terminal, and an arbitrary TSO user. This is controlled by the JS, JO, and MU parameters (see MP EXECUTION PARAMETERS).

9.3.6.1. JDSINIT initializes Journal Management. It issues this TSO command sequence:

```
IFREE JOURNAL
ALLOC F(JOURNAL) SYSOUT BLOCK(jbs) SPACE(pri,sec)
```

Where:

"jbs" is the JBS input parameter (see MP EXECUTION PARAMETERS).

"pri" is the JKT input parameter divided by 2.

"sec" is the JKT input parameter divided by 10.

9.3.6.2. JDSFIN finalizes Journal Management. It issues the TSO command:

```
FREE F(JOURNAL) SYSOUT(js)
```

Where "js" is the JS input parameter (see MP EXECUTION PARAMETERS).

9.3.6.3. JOURNAL accepts a character string to be written to the journal, and a halfword message type indicator. The message is written to the journal only if it has been initialized and enabled for that message type. If this brings the line counter to its threshold value, JDSFIN and JDSINIT will be called.

When a message of type ACTION_REQUIRED is written to the journal, it is also written to the system operator via the PL/I DISPLAY verb. But note that if this message type has been parametrically disabled (see MP EXECUTION PARAMETERS), no processing at all occurs.

When a message of type ALARM is written to the journal, it is written via DISPLAY, and JOURNAL executes SIGNAL ERROR. But note that if this message type has been parametrically disabled (see MP EXECUTION PARAMETERS), no processing at all occurs. This must not be done in the current implementation, because callers who use this feature to terminate MP abnormally do not now provide a control path for a RETURN from such a call to JOURNAL.

9.3.7. PARAMETER MANAGEMENT PACKAGE

The Parameter Management Package consists of entry GETPARM, called by the initialization section of the Main Control Module. It reads and decodes the parameter data set that may be supplied for each execution of MP. The parameter data is read from file PARMS, where it is formatted for a single GET DATA statement. Permitted names and defaults are given in MP EXECUTION PARAMETERS. GETPARM prints a copy of its input, and a list of all options in effect, using PL/I file SYSPRINT.

9.3.8. TIMER MANAGEMENT PACKAGE

The Timer Management package consists of Assembler-language routine TIMSRVS, with its three entry points.

9.3.8.1. STIMER

STIMER interfaces to the Host Operating System's "Set Interval Timer" services. It accepts these parameters:

* A time interval in hundredths of a second. If this is zero, the request is to cancel any outstanding time interval without POSTing. Otherwise, any new interval automatically cancels any old interval outstanding for the same task, again without POSTing.

* The address of an ECB. This will be cleared, and POSTed when the interval expires normally.

* The address of a seven-word work area that STIMER can consider his own until the interval expires or is cancelled.

9.3.8.2. TBEGIN

TBEGIN notes the begining of an interval to be measured by TEND. It accepts the address of a two-fullword work area which the caller agrees not to change until the matching call to TEND has been made.

9.3.8.3. TEND

TEND measures the real elapsed time since the matching call to TBEGIN. It accepts the same address as was passed to TBEGIN, and returns a fullword containing elapsed time in .01 seconds.

9.3.9. MISCELLANEOUS SERVICES

9.3.9.1. HEX ROUTINE

HEX accepts a pointer, a halfword byte count, and an optional halfword increment to be added to the pointer. It returns a varying character string which is the hexadecimal field representation of the memory bytes so indicated. Be sure that HEX is declared to return a string at least 1 greater than twice the maximum value of the byte count, as HEX does its UNPK right into the output string.

9.3.9.2. IDINIT ROUTINE

IDINIT accepts two 8-character strings and fills them with the Charge Number and TSO Userid under which it finds itself running.

9.3.9.3. NEGPTR ROUTINE

NEGPTR accepts a pointer and returns a pointer which is the arithmetic negative of the input. This is as required by PLOXI routine EXWAIT.

9.3.9.4. NOERR ROUTINE

NOERR turns off the PL/I STAE for debugging. It is documented elsewhere (see Ref. 15).

9.3.9.5. PERM ROUTINE

PERM produces all legal substituends for a character string containing "wild" characters. It accepts a varying character string which may contain up to seven question marks. It returns a substituted value of the same length. Each time PERM is called it will return a different value, and if the same caller calls it enough times sequentially, it will eventually return a duplicate of the first value returned. Only this tells the caller that his case is hopeless.

PERM is self-initializing and non-read-only. Whenever it finds a zero value already in its cumulative counter, it picks up a random value from the system timer. Whenever it generates a zero value, it skips it, advancing to 1 (otherwise, a caller might never duplicate his first received value).

9.3.9.6. SCANNER ROUTINE

SCANNER builds a table of SUBSTR parameters breaking a given data string down into simple subfields delimited according to a pattern string. The pattern consists of an odd number of characters, each "PAIR" of which consists of a "TYPE" ("N" for numeric fields, anything else for anything else) and a terminator (not present in the final "pair"). The output table has three columns. The first two are "SUBSTR" parameters and the third is the numeric equivalent of the substring (0 for non-"N" subfields). The table is filled in for as many entries as there are pattern "PAIRS", even if this means using zero length values in the final rows. Table overflow checks are not made -- this is the caller's problem.

SCANNER returns a Boolean (1-bit) value which is always True unless a non-numeric character was found in a numeric subfield. If the value is False, there is no indication of how many errors occurred, or where. The corresponding numeric equivalents are calculated as if invalid characters were compressed out ("3X2B1" -> 321, "TEN" -> 0). Null numeric fields are considered valid, and have the value zero. This routine uses masking to convert a numeric EBCDIC character to binary, which means it is EBCDIC-dependent.

9.3.9.7. TPUT AND TPUTUID

TPUT writes a character string out to the terminal. A second entry, TPUTUID, writes to the terminal of a specified TSO user. The program is documented elsewhere (see Ref. 16) and is not further covered here.

9.3.9.8. UNHEX ROUTINE

UNHEX accepts a character string not longer than 8 bytes, which had better contain nothing but legal hexadecimal field data. It returns a binary fullword equivalent. Invalid input results in invalid output, with no explicit notification of the fact.

9.3.9.9. XATOE ROUTINE

XATOE accepts a character string which it translates from ASCII to EBCDIC. It uses a copy of CCN standard table CCNTRATE.

9.3.9.10. XETOA ROUTINE

XETOA accepts a character string which it translates from EBCDIC to ASCII. It uses a copy of CCN standard table CCNTRETA; however, it alters the first byte of this copy so that EBCDIC "Null" (X'00') is treated like EBCDIC "blank" (X'40').

9.3.9.11. ACTSTAT ROUTINE

ACTSTAT acquires resource utilization statistics from the Host Operating System and formats a text string suitable for output via JOURNAL. Certain values are saved between calls to ACTSTAT, so that incremental statistics can also be produced. For this reason, the caller must pass a pointer to a nine-fullword work area, initialized to zeros, and not subsequently modified. ACTSTAT returns a varying character string containing captions and values for:

* Checkpoint number -- a cumulative counter.

* Real time, both cumulative and incremental.

* CPU time, both cumulative and incremental.

* I/O count, both cumulative and incremental.

* Swap count, both cumulative and incremental.

* Swap load, both cumulative and incremental.

* MUS, both cumulative and incremental (see Ref. 11).

* Total cost, both cumulative and incremental.

* The effective Region size.

9.4. MP EXECUTION FILE REQUIREMENTS

Assuming that you have the mechanisms for invoking and feeding MP as a TSO session, you must set up either a LOGON procedure or a command procedure to allocate needed files. You will need the following files:

* File IKJPUTL should be allocated the load library containing MP's special version of IKJPUTL and its aliases. This can be the same library that contains MP itself, or a different one. The file name IKJPUTL is not used for any other purpose.

* File PARMS should be allocated the data set containing the input parameters. If you have none, you can omit this file.

* File DSNFILE should be allocated a specific data set for each IP server executing MP, as it has a continuity function. This data set is described under JOB MANAGEMENT.

* File JOBFILE should be allocated a specific data set for each IP server executing MP, as it has a continuity function. This data set is described under JOB MANAGEMENT.

* File SYSPRINT should be allocated for the PLIX running system, and for GETPARM's option lists.

* File PLIDUMP should be allocated if you want a dump on ERROR conditions.

* A minimum of nine DD DYNAM's should be available. These are for FTS (5), GENERAL (1), the Journal (1), and various TSO Command Processors (2) called by MP.

9.5. MP EXECUTION PARAMETERS

Name	Type	Dflt	meaning
AV	CHAR(8)	'NSWP01'	The volume to be used for data set allocation in CAT ENTER NAME processing. WOE SUBMIT also requests this volume for the allocation of Standard System Output files.
BF	FIXED	1	The conversion factor between IP RPB and OS blocking factor. OS=IP*BF.
BL	FIXED	200	The Buffer Length to be used in allocating the MP universal buffer pool. This is expressed as the number of bytes in the DATA field of the maximum size IP message that can be contained in a buffer.
CN	CHAR(8)	' '	The Charge Number for submitting batch jobs. A blank value causes the Charge Number under which MP finds itself running to be used.
JBS	FIXED	508	The OS BLKSIZE for the Journal Data Set.
JKT	FIXED	1000	The maximum number of records to write to the Journal before scheduling it for printing.
JLR	FIXED	504	The OS LRECL for the VB Journal Data Set. If this is zero, V-format records will be produced, and LRECL will be set to JBS-4.
JMASK	BIT(16)	(16)'1'B	The Journal enabling mask. A 1 bit enables writing of the corresponding message type, and a 0 bit disables it. The message types currently defined, with their bit numbers, are: 0: STATISTICS 1: COMMENT 2: ALARM 3: TSO_COMMAND 4: TSO_RESPONSE 5: INCOMING_MSG 6: OUTGOING_MSG 7: INCOMING_DATA 8: INCOMING_BINARY 9: OUTGOING_DATA

			10: OUTGOING_BINARY 11: ACTION_REQUIRED
JO	BIT(1)	'0'B	Journal-Online switch. A 1 bit causes the journal to be written online, via TPUT. This bit is independent of JS and MU.
JS	CHAR(1)	'A'	Journal-SYSOUT class. A non-blank value causes the journal to be written to the indicated SYSOUT class. This is independent of JO and MU.
MJ	CHAR(8)	' '	The value of MYJOB in the Exchange window connecting MP with the network.
MT	CHAR(8)	' '	The value of MYTAG in the Exchange window connecting MP with the network. A blank value causes the USERID under which MP finds itself running to be used.
MU	CHAR(8)	' '	The Monitoring Userid. This represents a TSO Userid to which journal data will be transmitted as written, if and when the user is logged on. If it is blank, no such transmission occurs. This is independent of JS and JO.
NA	BIT(1)	'0'B	The No-ASCII switch. A 1 bit causes translation between EBCDIC and ASCII to be suppressed. This is used for local testing.
NB	FIXED	40	The number of internal message buffers to be contained in MP's universal buffer pool. Small numbers save core.
NE	BIT(1)	'0'B	The No-Error switch. A 1 bit suppresses PLI STAE processing. This is used for testing.
NF	FIXED	12	The number of asynchronous functions to activate. this is only used for testing.
NX	FIXED	10	The number of Exchange operations that may be pending on the output channel of the window connecting MP to the network. Large values effect a sort of 'blocking' on this channel.
OID	CHAR(8)	'OUTPTT'	The third-level index name to be used for MP Standard System Output data sets.

PW	CHAR(8)	' '	The Password to be used for submitting batch jobs.
QN	CHAR(8)	'NSWOUTPT'	The GMF Queue name to be used for notification from CCN's SPOOL3 output writer to MP.
SD	CHAR(8)	'W'	The SYSOUT Destination to be specified when submitting batch jobs.
TI	FIXED	5	The Timer Interval, in minutes, to govern the frequency of performing the periodic functions assigned the TIMER function.
YJ	CHAR(8)	'ARPA'	The value of YOURJOB in the exchange window connecting MP with the network.
YT	CHAR(8)	'IPTASK2'	The value of YOURTAG in the exchange window connecting MP with the network.

9.6. INSTALLING MP

Installing and maintaining MP involves various libraries and data sets. When running under MVT, MP will almost always have to be overlayed, at least to the point of keeping GETPARM and its GET DATA support out of the machine except during MP initialization.

9.6.1. DATA SETS

The following data sets are typically maintained as a part of MP:

* The Source Library contains the PL/I and Assembler-language source modules. Each of these is so designed as to produce an object deck terminated by a NAME card, and in some cases, other Linkage Editor control statements.

* The Compile-Time Library contains various %INCLUDE packets needed to compile PL/I routines. Note that no special Assembler-language macros are defined for MP.

* The Module Load Library contains load modules produced by individually compiling the members of the Source Library. It is redundant, and need not actually be kept, although it is a convenience in updating MP.

* The Final Load Library contains the executable copies of MP and IKJPUTL.

9.6.2. OVERLAY STRUCTURE

In desigining an overlay structure for MP, the important point is to isolate GETPARM and all its IBM library routines from the rest of the program. This represents about 14K of code which, once used, is worthless.

Technically, all the Asynchronous Functions can reside in exclusive segments. However, performance will be seriously degraded if EXIN, EXOUT, and FTS cannot operate without overlay activities. A suggested structure is:

* ROOT SEGMENT: All code not placed elsewhere.

* Segment 2: GETPARM and all its IBM support.

* Segment 3: EXIN, EXOUT, FTS, and all Exchange Management routines.

* Segment 4: GENERAL, OENSUB, NOTIFY, STATUS, OPR, MAKENOT, TIMER, and all Job Management routines except GMOPEN.

Modules that must reside in the root segment of any program in which they are used include GMOPEN and PERM.

Note that it is necessary to leave reference IHEERRA unresolved.

An easy way to ensure that all common areas and files are in the root segment is to include the output of compiling COMMONS. You can use a REPLACE to get rid of **DUMMY1, **DUMMY2, PLISTART, and PLIMAIN in the module.

The following STATIC EXTERNAL control sections will then reside in the root segment:

* COMMON1 - The Master common block;

* COMNOTE - NOTIFY's static data;

* COMSTAT - STATUS' static data;

* COMEX - Static data for both EXIN and EXOUT;

* COMGEN - GENERAL's static data;

* COMFTS - Static data for all five FTS functions;

* COMOPR - OPR's static data;

* COMTIM - TIMER's static data;

* ERRIDS - standard error codes referenced by most callers of COMMAND.

9.6.3. MP MODULE STRUCTURES

Module	Lang	Entries	Extrefs	Commons	TSO Commands
ACTSTAT	ASM	ACTSTAT			
COMEXEC	ASM	COMEXEC COMFIN COMINIT	IKJGETL IKJPTGT IKJPUTL		
COMMAND	PLI	COMMAND	COMEXEC JOURNAL	COMMON1	
COMMONS	PLI			COMEX COMFTS COMGEN COMMON1 COMNOTE COMOPR COMSTAT COMTIM ERRIDS	
DSNHIST	PLI	HISTIN	HISTOUT	COMMON1	
COPYQDEQ	PLI	COPYQ ENQ NDEQ	JOURNAL NEGPTR XQTOE XETOA	COMMON1	
EXDONE	PLI	EXDONE		COMMON1	
EXIN	PLI	EXIN RSEXI	COPYQ ENQ EXCH EXDONE JOURNAL NDEQ XATOE	COMEX COMFTS COMGEN COMMON1 COMSTAT	
EXOUT	PLI	EXOUT RSEXO	COPYQ ENQ EXCH EXDONE EXWAIT NDEQ	COMEX COMFTS COMMON1	

Module	Lang	Entries	Extrefs	Commons	TSO Commands
FTS	PLI	FTS	COMMAND	COMFTS	ALLOC
		RSFTS	COPYQ	COMMON1	DELETE
			ENQ		FREE
			HISTIN		IFREE
			HISTOUT		RELEASE
			JOURNAL		
			LOOKUP		
			MAKEF		
			NDEQ		
			OUTDSN		
			SCANNER		
			TBEGIN		
			TEND		
GENERAL	PLI	GENERAL	COMMAND	COMFTS	ALLOC
		RSGEN	COPYQ	COMGEN	ATTRIB
			ENQ	COMMON1	CANCEL
			JOURNAL	COMSTAT	DELETE
			JTDEL		RFREE
			LOOKUP		IFREE
			MAKEE		RENAME
			NDEQ		
			OENSUB		
			OUTDSN		
			PERM		
			SCANNER		
GETPARM	PLI	GETPARM		COMEX	
				COMFTS	
				COMGEN	
				COMMON1	
				COMNOTE	
				COMOPR	
				COMSTAT	
				COMTIM	
				ERRIDS	
HEX	ASM	HEX			
IDINIT	ASM	IDINIT			
IKJPUTL	ASM	IKJGETL			
		IKJPTGT			
		IKJPUTL			
JOBMGT	PLI	JTADD	COMMAND	COMMON1	DELETE
		JTDEL	ENQ	COMSTAT	
		JTREAD	JOURNAL		
		MAKENOT	UNHEX		
		OUTDSN			

Module	Lang	Entries	Extrefs	Commons	TSO Commands
JOURNAL	PLI	JDSFIN	COMMAND	COMMON1	ALLOC
		JDSINIT	TPUT		FREE
		JOURNAL	TPUTUID	FREE	
LOOKUP	ASM	LOOKUP			
MAINPGM	PLI	MAINPGM	COMFIN	COMEX	
			COMINIT	COMFTS	
			ENQ	COMMON1	
			EXIN		
			EXCLOSE		
			EXOPEN		
			EXOUT		
			EXWAIT		
			FTS		
			GENERAL		
			GETPARM		
			IDINIT		
			JDSFIN		
			JDSINIT		
			JOURNAL		
			NOERR		
			NOTIFY		
			OPR		
			PLIDUMP		
			STATUS		
			RSEXI		
			RSEXO		
			RSFTS		
			RSGEN		
			RSNOT		
			RSOPR		
			RSTAT		
			RSTIM		
			TIMER		
MAKEF	ASM	MAKEF			
		MAKEE			
NEGPTR	ASM	NEGPTR			
NOTIFY	PLI	NOTIFY	JOURNAL	COMMON1	
		RSNOT	JTADD	COMNOTE	
		GMCLOSE			
		GMQUEUE			
		GMOPEN			
		GMUSER			
		MAKENOT			
		NEGPTR			
		NDEQ			
		UNHEX			

Module	Lang	Entries	Extrefs	Commons	TSO Commands
OENSUB	ASM	OENSUB			
OPR	PLI	OPR RSOPR	NDEQ	COMMON1 COMOPR	
PERM	ASM	PERM			
SCANNER	PLI	SCANNER			
STATUS	PLI	RSSTAT STATUS	COPYEQ ENQ HEX INTMT JTDEL JTREAD MAKENOT NDEQ SCANNER TMTSTAT	COMMON1 COMSTAT	
TIMER	PLI	RSTIM TIMER	JDSFIN JDSINIT JOURNAL NEGPTR STIMER	COMMON1 COMNOTE COMTIM	
TIMSRVS	ASM	STIMER TBEGIN TEND			
TMTSTAT	ASM	INTMT TMTSTAT			
UNHEX	ASM	UNHEX			
XLATE	ASM	XATOE XETOA	CCNTRATE CCNTRETA		

END